

Management of Cryptographic Identities using DNS-based Authentication of Named Entities (DANE)

Bachelor thesis

Sebastian Simon

Matriculation number: 375871

Technische Universität Berlin



Department for Security in Telecommunications (SecT)

First examiner: Prof. Dr. Jean-Pierre Seifert

Second examiner: Prof. Dr. Stefan Schmid

Supervisor: Nils Wisiol

Berlin, 2023-04-25

Abstract

The Domain Name System (DNS) provides a service to look up server addresses 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 automating and simplifying deployment of DNS-based Authentication of Named Entities (DANE), a set of security protocols which can be used by clients to authenticate a server. In particular, we establish a prototype of an identity management application, driven by user-centered design, which helps in DNS zone administration with automatic synchronization between a TLS certificate on the server and a TLSA Resource Record in the DNS. This synchronization provides the TLS certificate association which enables clients to cross-check cryptographic data, such as public keys or certificates, from the server itself and the DNS. TLS certificates are generally issued by external Certificate Authorities; the purpose of this DANE binding is to be able to avoid trust in them.

The goal of the application is to minimize human error and increase security. However, implementing DANE requires careful management; an implementation needs to obey different requirements and support different scenarios. We examine the life cycle events of a server identity and review the relevant specifications and standards, all of which help ensure correct management.

Zusammenfassung

Das *Domain Name System* (DNS) ist ein Dienst zum Abrufen von Serveradressen anhand eines Namens, aber standardmäßig fehlt in diesem Dienst eine Authentifizierung von Daten und diese sind vor einer Verfälschung nicht geschützt. Unterschiedliche Sicherheitsprotokolle und -erweiterungen existieren, die jedoch eine hohe Komplexität haben. Daher schreitet deren Einsatz nur langsam voran.

Diese Arbeit befasst sich mit einer automatisierten und vereinfachten Bereitstellung der DNS-basierten Authentifizierung von Benannten Entitäten (*DNS-based Authentication of Named Entities* – DANE), einer Reihe von Sicherheitsprotokollen die ein Client benutzen kann, um einen Server zu authentifizieren. Wir errichten konkret einen Prototypen einer Anwendung für Identitätenmanagement mit einem benutzerzentrierten Design; diese soll bei der DNS-Zonen-Administration in Form von einer automatischen Synchronisierung zwischen einem TLS-Zertifikat eines Servers und einem TLSA-Eintrag im DNS helfen. Diese Synchronisierung stellt die TLS-Zertifikats-Assoziation bereit, mit der ein Client kryptographische Daten wie zum Beispiel öffentliche Schlüssel oder Zertifikate vom Server selbst und vom DNS abgleichen kann. Generell werden TLS-Zertifikate von externen *Certificate Authorities* ausgestellt; Zweck dieser Ausprägung von DANE ist es, auf das Vertrauen in diese verzichten zu können.

Ziel der Anwendung ist es, menschliche Fehler auf ein Minimum zu beschränken und die Sicherheit zu erhöhen. Allerdings erfordert eine Implementierung von DANE sorgfältiges Management; sie muss unterschiedlichen Anforderungen gerecht werden und unterschiedliche Szenarien unterstützen. Wir betrachten die Ereignisse im Lebenszyklus einer Serveridentität und begutachten relevante Spezifikationen und Normen, die beim korrekten Management behilflich sind.

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

Place, Date:

Berlin, 2023-04-25

Signature:

Sebastian Simon

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 5 |
| 2 | Background | 8 |
| 2.1 | History of the Internet, DNS, and the emergence of DNSSEC | 8 |
| 2.2 | Trust in TLS Certificate Authorities | 10 |
| 2.3 | Deployment of DANE | 10 |
| 3 | Protocols involved in DANE identity management | 13 |
| 3.1 | Transport Layer Security (TLS) certificates | 13 |
| 3.2 | The Domain Name System (DNS) | 17 |
| 3.3 | DNS Security Extensions (DNSSEC) | 19 |
| 3.4 | The DANE TLS association (TLSA) | 22 |
| 3.5 | Other DNS- and TLS-related security protocols | 34 |
| 4 | The standards aspect of DANE identity management | 37 |
| 4.1 | Terminology | 37 |
| 4.2 | Identity life cycle and identity management | 42 |
| 5 | User-centered design implementation using web technologies | 50 |
| 5.1 | Tools | 50 |
| 5.2 | Back end | 53 |
| 5.3 | Web application and user interface | 55 |
| 6 | Conclusion and future work | 58 |
| | References | 59 |
| | Appendix | 68 |
| | Code samples | 68 |
| | Design wireframes | 71 |

1 Introduction

The Internet has come a long way since its initial conception in the late 1960s and its rapidly increasing adoption by the wider public in the following decades. However, many fundamental protocols—especially in its early days—have been engineered with only few security considerations in mind. Over time, many different protocols, standards, and practices have been proposed and standardized in order to improve security of different systems, particularly for the Internet.

Nowadays, when a web browser loads a website, the underlying connection is almost always secure: our data is almost completely safe from eavesdropping and tampering, and will arrive at its intended destination in a timely manner most of the time. In other words, the security-related properties of *confidentiality*, *integrity*, *availability*, and *authenticity* hold in most cases today.

Typically, when a web browser connects to the Internet *today*, e.g. requesting a resource, it first queries the Domain Name System (DNS), the “*phone book*” of the Internet. The original purpose of the DNS is to provide a mapping between human-friendly names, such as `www.example.com`, and machine-friendly IP addresses, such as `93.184.216.34`. IP addresses are stored in a Resource Record (RR) of type A (for IPv4) or AAAA (for IPv6). Other RR types are also needed, e.g. the name server (type NS) which provides authoritative data on the corresponding administrative portion of the DNS (the *DNS zone*), but the DNS can also store arbitrary information [1]. RRs are identified by a DNS name; RRs of the same name and type form a Resource Record Set (RRSet). If supported, the response from the DNS is then validated using the DNS Security Extension protocol (DNSSEC) [2]. Once the response is received, the router routes to the obtained IP address, eventually reaching a server.

The browser requests the Transport Layer Security (TLS) certificate from a *TLS server* during the so-called *TLS handshake*. During this client-initiated process, the TLS version is negotiated, and a stateful connection (e.g. via TCP sockets) between client (i.e. the browser) and server is established, over which encrypted data is sent. Web pages are then served over the Hypertext Transport Protocol (HTTP), the *secure* version of which is called HTTPS.

All of this happens in the background. The only prominent indicator for the average Web user is a little lock icon next to the “address bar”, which contains a URL starting with `https://`.

A TLS certificate needs to be cryptographically *signed* by some issuer in order to be trusted; this also implies trust in that issuer. Certificates signed by other certificates form a *certificate chain* [3]. Web browsers come with a *trust store* pre-installed, which contains a plethora of existing certificates owned by Certificate Authorities (CAs). These are *root certificates* and *intermediate certificates*, and are collectively called *trust anchor certificates*. Trust anchor (TA) certificates have been used to sign

nearly every website certificate that the web browser will come across. These website certificates are also called *end-entity* (EE) certificates.

In this particular Public Key Infrastructure, referred to as “PKIX”,¹ the trust store serves as a “secondary source” that the browser can consult in order to “cross-check” a given certificate chain. However, these trust anchor certificates are not safe from security vulnerabilities; a compromised trust anchor (or CA) might compromise the security of a TLS connection. One could consider using one’s own trust anchor certificate or a self-signed end-entity certificate, but currently, such custom certificates are not configured to be trusted by every browser, resulting in the TLS connection being rejected by the browser.

An alternative approach is to place certificates into the DNS, which in turn serves as a secondary source. This is the idea of DNS-based Authentication of Named Entities (DANE), operating in the realm between DNS and target server. DANE is used to authenticate a target server by verifying if its cryptographic data matches with the cryptographic data found in the corresponding DNS zone, based on its DNS name. In the case of TLS, a RR of type TLSA is used. A validating client will query this RR and compare it to the TLS certificate, so a DNS administrator will need to make sure to put a TLSA RR with the relevant certificate data in their DNS zone [4].

But this is easier said than done. Manually publishing a TLSA RR requires regularly updating the DNS zone, using tools to convert the certificate into different formats, extract specific information from it, compute digests using hashing algorithms, and keeping track of which RRs to keep and which ones to delete—every step being prone to human error, potentially compromising security.

This process can be simplified using automation, but even this requires careful management. On top of that, there are several requirements for both clients and servers that need to be covered, and different validation scenarios to be supported. This is what this thesis focuses on. We establish a prototype of a DANE identity management application that synchronizes TLSA RRs with TLS certificates automatically. The approach used to ensure correct management and validity is to consider *server identities* and examine their entire *identity life cycle*, i.e. all states and transitions relevant to DANE RR management.

To illustrate the relationships between the objects involved in DANE management, figure 1.1 provides a complete entity-relationship model. It helps visualize which parts exactly are affected by the proposed identity manager, and helps decide how objects need to be manipulated. The left half (including “TLSA RR”) is the DNS side, the right half is the Web service side. The subsequent sections will refer back to this diagram, pointing out specific relationships. There are five links that can have a cardinality of zero; if and only if all of them are exactly zero in a specific system, it means that this system *does not support* the TLSA binding of DANE.

Section 2 briefly examines the *history and prior art* on this topic. Section 3 explores the *protocols and specifications* and section 4 explores the more general *standards* involved in DANE identity management. Section 5 explores the *implementation* aspect of the proposed application. Section 6 reviews the efforts of establishing DANE identity management done here and provides some pointers for future work.

¹The “X” in “PKIX” stands for the corresponding ITU-T standard, “X.509”.

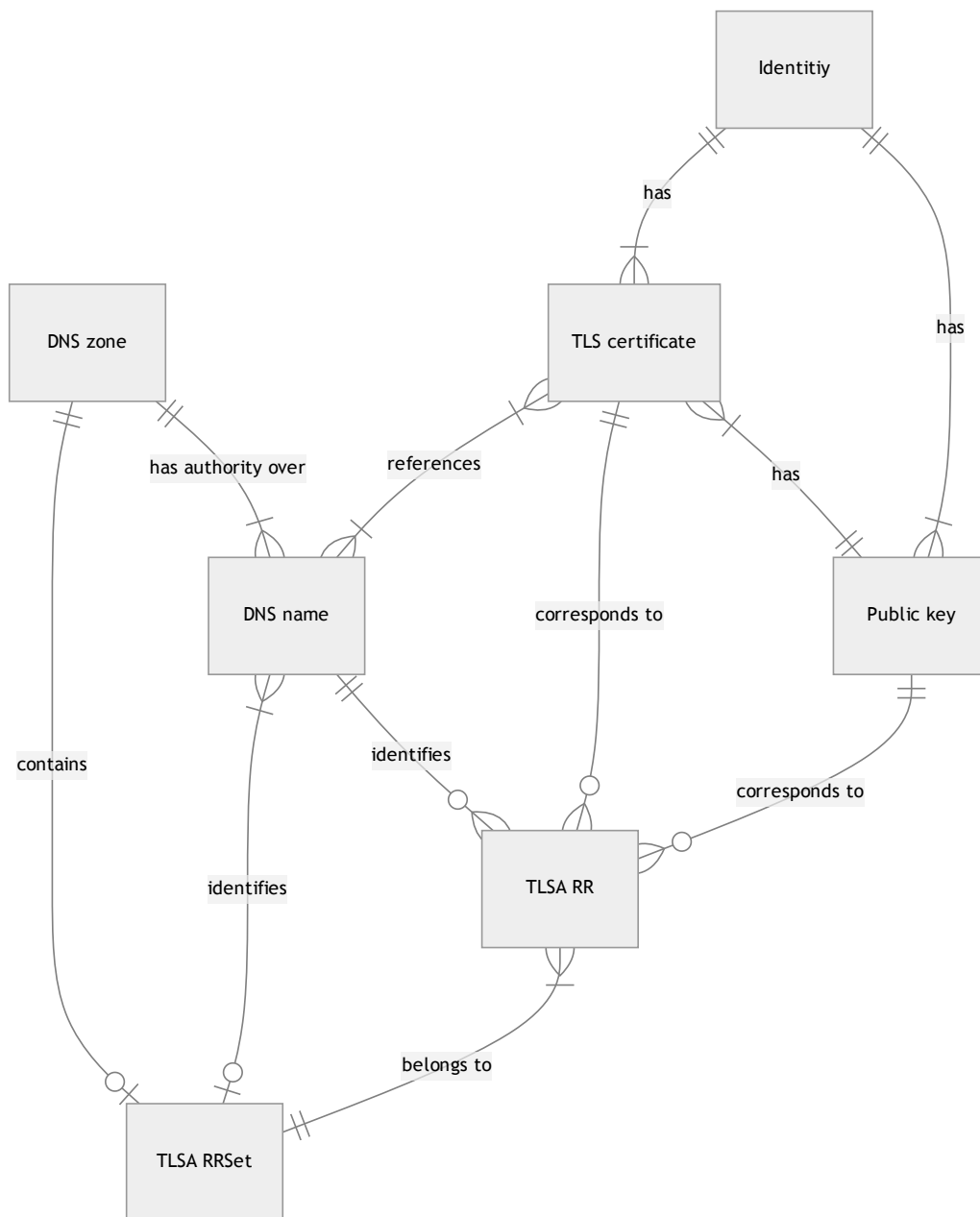


Figure 1.1: Relationship between entities concerning DNS, TLS certificates, and DANE

2 Background

DANE is a relatively new set of protocols, but in order to understand its origin and purpose we need to understand its place in history. This section covers the history and security-related developments in the DNS and in the TLS protocol, including issues in DNSSEC deployment, and vulnerabilities and the proposed mitigations and solutions. Finally, the state of the art in the employment of the DANE binding “TLSA” will be covered, which rests upon DNS (and DNSSEC) and TLS.

2.1 History of the Internet, DNS, and the emergence of DNSSEC

World-wide communication between machines—what eventually became *the Internet*—has a rich history. Briefly, remote hosts are accessible by an *address*, but also by a *host name*; the mapping between the two used to be managed in centralized databases within different internets until the early 1980s. Due to the rapidly growing size of internets, the diversity in their operation, and administrative and personnel issues, this quickly became unmanageable, thus a decentralized database needed to be established which delegates administration to separate, hierarchically structured *domains*. This system is known as the *Domain Name System*, or DNS, today [1]. See section 3.2 for a more in-depth overview of DNS.

Many threats and vulnerabilities have been identified in the DNS [5]. One of the major vulnerabilities is known as *DNS cache poisoning*¹. DNS name servers utilize caches to store query results in order to improve performance. The exploit happens when an attacker taints the DNS cache by bombarding DNS servers with DNS query *responses* until one server decides to make a DNS query and inadvertently accepts one of the attacker’s responses instead of the real one. This problem has historically been mitigated by using randomized transaction IDs as well as randomized destination ports to avoid guessing them easily [6].

But the long-term solution² to this problem are the DNS Security Extensions (DNSSEC), a protocol proposed in 1997 [5]. DNSSEC introduces a public-key infrastructure into the DNS which publishes a public key of the DNS zone, whose private counterpart is used to sign the RRSets within the zone. Additionally, a digest of the public key is also provided and maintained by the *parent* zone, which in turn has its own public key and signatures. This additional information is added to every DNS response. Section 3.3 explains how DNSSEC works in detail.

¹*DNS cache poisoning* is also known as *DNS spoofing* or *name chaining attack*.

²However, the algorithms utilized by DNSSEC would need to be fundamentally changed and readied for *post-quantum cryptography*, i.e. still provide security in the presence of the rapidly evolving field of quantum computing which threatens to “break” even the most advanced cryptographic algorithms commonly used today [7].

After nearly a decade of design work, the DNSSEC standard was published in March 2005 [2]. In August 2004, while DNSSEC was still in development, RFC 3833 analyzed the identified threats and vulnerabilities and evaluated how the DNSSEC standard would solve these problems [5]. The security properties achieved by DNSSEC are data integrity and data origin authentication. DNSSEC is not concerned with achieving confidentiality. DNSSEC validation is also not always applicable in securing every part of DNS communication. Section 3.5 provides an overview of other DNS-related security protocols, which solve these issues.

Another security concern that remains with DNSSEC is *(Distributed) Denial of Service* (DDoS, DoS)³, impacting availability of hosts that handle DNS queries. In fact, the problem is even exacerbated with DNSSEC due to the larger packet payload. The availability concern will not be discussed in detail in this thesis, but some recent efforts to mitigate DDoS attacks have been done with Software Defined Networks (SDNs) [8], [9].

Starting in December 2009, DNSSEC was on its way to be launched for the root DNS zone. This was a lengthy process that involved a lot of testing, e.g. the effects of the larger payloads in DNS responses, and was finally completed in July 2010: for the first time, root name servers would serve a signed root zone. The process involved two lengthy ceremonies of generating public and private key pairs and using them to sign the root zone [10].

But this is not the end of the story. In the years after the launch, DNSSEC struggled to find adoption; only very few domain registrars offered support for DNSSEC, and not all of them deployed it correctly or fully, even in 2017 [11], [12]. Since correct deployment of DNSSEC is complex and a misconfiguration threatens the availability of the domain, it might explain the reluctant pace of adoption. Common misconfigurations include the mismatch of keys, digests of keys, and signatures, the lack of one or more required RRs, or the missing renewal of expired keys or signatures [13].

The key roll-over of the root zone was another issue that took eight years to plan and complete. The fact that so many complex steps were involved in the design and rollout of DNSSEC, taking two decades of work, likely was another reason why DNSSEC deployment remained low before 2018. As Dan York, leader of the *Internet Society's Open Standards Everywhere* project, puts it [14]:

Deployment is moving on. I think there was a pause between 2015 and 2018, while we waited around for the changing of the root key, where people running the DNS infrastructure kind of wanted to wait and see how the root key rollover would go. It completed in 2018 and all things are good, the lights are green, and now we're seeing in the charts how DNSSEC deployment is going up.

After the first successful key roll-over, DNSSEC adoption has been increasing. Key roll-overs became a quarterly public event called the “Root KSK Ceremonies” [15]. In 2019, registrars have already noticeably increased their support for DNSSEC [16].

By the end of 2020, the *Internet Corporation for Assigned Names and Numbers* (ICANN) has announced that *all* generic top-level domains (*gTLDs*, such as *com*, *org*, *info*, etc.) support DNSSEC [17]. Today, 142 of the 248 country-code top-level domains (*ccTLDs*, such as *de*, *us*, *ua*, etc.),

³In the context of DNS, specifically, a variant of this exists, known as a *DNS amplification attack*.

i.e. 57.3 %, support DNSSEC [18]. However, not many domains *themselves* deploy DNSSEC: only an estimated 5.9 % of .com domains, 8.4 % of .edu domains, and—at least—83.7 % of .gov domains fully deploy DNSSEC [19]. Almost a third of the world performs DNSSEC *validation*: 31.7 % worldwide on average, the most in Europe (46.2 %), the least in Asia (27.6 %), with Oceania, North and South America, and Africa falling in between [20].

2.2 Trust in TLS Certificate Authorities

TLS certificates are signed by Certificate Authorities (CAs). See section 3.1 for more details on how TLS works.

There are hundreds of root certificates owned by over 100 different CAs [21]. Since every root certificate (and intermediate certificate) can issue certificates for any domain, there is a risk of forged certificates being issued for popular domains, either by a malicious CA or by an attacker targeting a CA. This can result in the encryption provided by TLS to be circumvented [22].

Quite a few of these attempts have been documented; these are just two examples:

- In 2011, fraudulent certificates have been issued for *Google* by *DigiNotar* after being hacked by Irani attackers; shortly afterwards, the CA declared bankruptcy [23]–[25].
- In 2015, this happened to *Google* again, this time issued by the Chinese registrar *CNNIC* via an intermediate certificate owned by *MCS Holdings*; the certificate’s private key might have been deliberately used to perform man-in-the-middle attacks [26].

A few solutions have been used to recognize and deal with these threats, including *certificate pinning* (or HTTP Public Key Pinning – HPKP), which is a protocol for accepting only a specific set of certificates or public keys, and *Certificate Transparency* (CT), which proposes a public append-only ledger for auditing issued certificates. In 2016, *Facebook* discovered certificates unexpectedly issued by *Let’s Encrypt*, which were requested by the hosting vendor managing one of *Facebook*’s domains; this was not malicious activity, but CT helped with quickly discovering these certificates as a possible security anomaly [25]. HPKP has been deprecated in 2017 due to its high complexity and potential for incorrect usage [27], [28].

Another challenge with certificates is their revocation. Two protocols have been in common use: the *Certificate Revocation List* (CRL) and the newer *Online Certificate Status Protocol* (OCSP), both explained in a bit more detail in section 3.1. Both of these require communication with a third party when issuing a revocation, as well as trust in that third party, and consultation of it when verifying the validity of a certificate. This means, there are privacy concerns and operational difficulties.

2.3 Deployment of DANE

Given the background in DNS and TLS and their vulnerabilities, the need for DANE as an alternative to Certificate Authorities is evident. See section 3.4 for a detailed look into DANE (TLSA).

DANE depends on correct DNSSEC deployment, but DNSSEC still has low usage and high complexity. DANE introduces *even higher* complexity and its usage is expected to be *even lower*.

The current deployment status of DANE remains low. In 2017, a handful of websites and mail servers with correct deployment of a TLSA RR have been listed as known examples [29]. DANE appears to be more popular in email transmission using SMTP with TLS, yet only a handful of email service providers support DANE correctly; the quantity of correct TLSA RR deployment also remains low, and is increasing only slowly [14], [30]. In 2020, just over 5,000 mail exchange servers have had DANE RRs deployed. By 2023, the number grew to over 9,000. A total of approximately 3,76 mil. DANE-protected SMTP domains have been recorded [31].

A set of common mistakes has been identified when deploying a TLSA RR, most of which come down to nonconformance with the relevant specifications [32]. Since DANE depends on DNSSEC, some of these are similar to the common misconfigurations of DNSSEC—e.g. only partial implementation—and some of them are *caused* by them. Lack of (reliable) automation and correct synchronization between TLS certificate and TLSA entry is a major issue. Another interesting point is the misguided deployment of DANE simply for the sake of using a *cool, new protocol*. DANE is not a one-and-done setup, but requires continual maintenance in order to ensure availability and integrity. Additionally, proper risk assessment and prioritization are important steps in securing any system [33].

As for the implementation efforts of DANE in DNS management systems, whereas some do support the TLSA RR in a DNS zone, *automatically managing* them still comes down to the individual DNS administrator. deSEC refers to third-party tools to create TLSA RRs, but is “*planning to provide a tool that is connected directly to [their] API in the future*” [34]; the corresponding GitHub repository includes designs for a DANE identity manager in an early stage of development [35]. Other popular domain registrars and Web hosting services, such as GoDaddy [36], netcup [37], cPanel [38], or CloudFlare [39], either do not have any plans on automatically supporting DANE whatsoever, or it is not a near-future priority on their road map, currently.

A few online tools exist to generate DANE RRs [40], as well as validators that test for correct configuration of TLSA RRs and display some debugging output [41], [42]. Software tools for these purposes exist, too [43], [44]. There is, however, a lack of *integration* of the services that these tools provide into applications that people use daily.

The DANE binding for TLS certificates for HTTP ultimately requires DANE validation in the browser. Some browser vendors have made some initial efforts in testing DANE validation but have hit certain roadblocks, e.g. high complexity, low performance, or TLSA lookups being blocked by the system [14], [30], [45]. Unfortunately, major browser vendors do not consider the DANE TLSA protocol to be of high priority, currently; it is unlikely that we will see built-in validation any time soon.

Mozilla Firefox has long-standing *Bugzilla* entries with requests to support DANE natively in the browser [46], [47]. These *bugs* acknowledge the chicken-and-egg problem of DANE: not enough Internet services deploy DANE—and DNSSEC for that matter—for implementers of client validators to care, and not enough client validation implementations exist for Internet services maintainers to care. They also chronicle the political environment surrounding the CAs *vs.* DANE debate. For

example, starting 2014, a lot of effort has been put into *Let's Encrypt*, a free CA. *Mozilla* is involved in the *Internet Security Research Group* (ISRG) responsible for facilitating the infrastructure of *Let's Encrypt* [48]. As of now, no work is actively being done to support DANE validation natively in *Mozilla Firefox*. However, a browser extension exists that promises to provide a built-in DANE validator using *DNS over HTTPS* [49].

The *Chromium* browser (and, by extension, *Google Chrome*) tracks its own *issues* [50], [51]. Other security protocols and approaches were more relevant at the time these issues were filed [52]. As of today, there are still no plans to support DANE on their road map.

Several arguments against DANE and DNSSEC have been presented, which focus on the flaws of DNSSEC itself, but also argue that DANE would not solve the problems of existing CAs, and instead would simply introduce *yet another* “CA”, only that it would be *government-controlled*. It is pointed out that DANE would make things worse because suspicious activity at a trust anchor would no longer affect the trust in a CA company, but in an entire DNS zone, e.g. for the top-level domain com [53], [54]. Complete DANE validation is computationally expensive and introduces a larger network load, thus increasing latency, which is cited as another reason why browsers have been avoiding its implementation [30], [45].

In summary, the current state of DANE is enveloped in a very long history of the life and death of cryptographic algorithms and protocols. The state of cryptography as a whole is ever-changing, and DANE is only one of many ideas being proposed, none of which are perfect. Only very few people see a priority in adopting DANE, so it is worth a try to improve the situation by taking steps towards simplifying the process—be it only for the motivation of more open-ended discussion about generally improving the trust model within communication systems.

3 Protocols involved in DANE identity management

This section summarizes the protocol specifications for TLS, DNS, DNSSEC, and DANE TLSA. It also mentions the other DANE bindings: IPSECKEY, SSHFP, OPENPGPKEY, SMIMEA. This provides the theoretical background for a technical implementation of DANE. Finally, some related protocols are briefly summarized and their relationships visualized.

3.1 Transport Layer Security (TLS) certificates

TLS *public-key certificates* are standardized by ITU-T X.509 since 1988 [3]. This subsection introduces the TLS protocol where it is relevant in its involvement as a DANE binding. The *Transport Layer Security* protocol—also known by its former name *Secure Sockets Layer* (SSL)—is the public-key infrastructure prevalent on the Web. By definition, a *public-key certificate* binds a *public key* to an *identity* [4], [55] (or simply a *name* [4] or *identifier* [56]). Section 4.1 covers the terminology around the word “identity”. According to the standard, a TLS certificate includes a set of names (server hostnames), some identifying information, such as a geographic address, a public key, a validity period (valid from *X* until *Y*), a signature, and a few other pieces of data.

This is modeled (see figure 1.1) by the following relationships:

- One *TLS certificate* belongs to one *Identity*
- One *TLS certificate* has one *Public key*
- One *Identity* can have multiple *Public keys* and multiple *TLS certificates*
- One *Public key* can belong to different *TLS certificates* (owned by the same *Identity*)
- One *TLS certificate* references one or more *DNS names* (or “host names”, “subdomains”, or just “names”)
- One *DNS name* can be referenced by multiple *TLS certificates*

Overview

A *public key* corresponds to a *private key*; both are mathematically “linked” with the property that encryption of data with *one* of the keys can only be reversed (i.e. the *ciphertext* being decrypted) with the *other* key. This idea is known as asymmetric cryptography, with *RSA* (named after its inventors, Rivest, Shamir, and Adleman) being the commonly used algorithm [57]. The public key within the certificate is used for encryption of data, whereas the private key (kept secret) can be used

for signing data. This means a certificate allows communication with *confidentiality*, *integrity*, and *authenticity*, which is the purpose of Web browsers requesting TLS certificates from TLS servers.

Certificates come in a few different formats and can be stored in files with different extensions depending on the format [55]. These are the two most common ones:

| Format | File extensions | Encoding and structure | Standard |
|---|------------------|---|-----------------------------------|
| DER (<i>Distinguished Encoding Rules</i>) | .crt, .cer, .der | Compact binary format of certificate data | ITU-T X.690 / ISO/IEC 8825-1 [58] |
| PEM (<i>Privacy-enhanced Electronic Mail</i>) | .pem | Base64 encoding of DER certificate | RFC 7468 [59] |

In order to prove the identity of the certificate holder, the certificate includes a signature. Simply put, a signature is *ciphertext* (i.e. the result of encryption), generated via a private key, of the hashed certificate data. If the identity behind the certificate uses its own private key corresponding to the public key found in the certificate, then this certificate is *self-signed*. On its own, a self-signed certificate doesn't confidently prove its corresponding identity. A website, therefore, usually has its certificate signed via the private key of another certificate; that certificate belongs to a *Certificate Authority* (CA) which is an entity that serves as the *issuer* of the website's certificate [60], [61]. Usually, there is a so-called *certificate chain* (or *certification path*) of three certificates served by any TLS server found on the Internet:

- The website's certificate is the *end-entity certificate* and is signed by an *intermediate certificate*;
- the *intermediate certificate* is a *trust anchor certificate* and is signed by a *root certificate*;
- the *root certificate* (also a *trust anchor certificate*) is self-signed.

The hierarchy involves an intermediate certificate to mitigate the risk of key compromise of that trust anchor. The private key of a root certificate is kept physically secure and is never accessed for any signing activity, except the *very few* times it needs to sign an intermediate certificate, whereas the intermediate certificates constantly need access to their private keys in order to sign end-entity certificates [61].

The trust model that involves this chain of certificates vouching for one another is referred to as a *public-key infrastructure* (PKI); this specific one, using the X.509 standard, is called PKIX. The root certificate itself is trusted by virtue of being pre-installed on the operating system within the *trust store*, a regularly updated set of established root CA certificates. The intermediate certificates are usually also pre-installed.

Workflow from issuance to secure connection

The workflow of *creating* a certificate and *verifying* its signature can be explained by the following small example [61]. In the appendix, a code sample using *OpenSSL*, an open-source cryptography library, is presented, which generates certificates, verifies them, and uses them to encrypt data.

There are three subjects involved: Alice (*she / her*), Bob (*he / him*), and a certificate authority (*the CA*). We start with the CA creating its certificate:

1. The CA generates a private key and a public key.
2. The CA generates a certificate file which includes its public key, its identifying information, a validity period, a serial number, and a signature generated by the private key.

The CA's certificate is (eventually or already) installed onto Alice's and Bob's systems.

Next, the subjects seeking to communicate with confidentiality and authenticity, Alice and Bob, create their own certificates, signed by the CA:

3. Alice and Bob each generate their own private key and public key.
4. Alice and Bob each create a *certificate signing request* file (CSR) which include their individual public keys as well as their individual identifying information.
5. Alice and Bob each send their individual CSR to the CA.
6. The CA creates one certificate for each CSR by attaching a signature, a validity period, and a serial number to it; the signature is generated by hashing and encrypting the CSR data using the CA's own private key.
7. The CA sends Alice's certificate to Alice, and Bob's certificate to Bob.

At no point did any private key get shared. Also, whether a certificate is an end-entity certificate or a trust anchor certificate is part of the certificate data itself (within the `basicConstraints.cA` extension). An end-entity certificate cannot issue another certificate.

If Alice now wishes to communicate with Bob, they'd have to share their certificates, and verify their signatures first:

8. Alice verifies Bob's certificate by hashing his certificate data and decrypting his signature using the CA's public key; if both resulting digests match, the verification is successful.
9. Alice extracts Bob's public key from his certificate.

If Alice publishes a signature, everyone can verify it using her public key, but only she could have created it using her own private key. Everyone can send Bob an encrypted message using his public key, but only he can decipher it using his own private key. The signature provides *authenticity* (and *integrity*), the encryption provides *confidentiality*—both are combined in a TLS connection [62].

In practice, arbitrary data is not directly encrypted using *public-key cryptography* due to performance reasons, but more importantly due to the nature of the RSA algorithm: only relatively short messages can be encrypted. Instead, a different algorithm (usually the *Advanced Encryption Standard*—AES) is used to generate a *symmetric* key. The symmetric key *itself* is encrypted using RSA and can then be shared.

10. Alice generates a symmetric key.
11. Alice encrypts the symmetric key using Bob's public key.
12. Alice hashes the symmetric key and encrypts the digest using her own private key, resulting in a signature.

13. Alice sends Bob both the encrypted symmetric key as well as the signature.
14. Bob decrypts the encrypted symmetric key using his own private key.
15. Bob verifies the signature using Alice's public key (i.e. the digest of the symmetric key must match with the decrypted signature).

At this point, the symmetric key has been shared, which can now be used by *both* subjects to both encrypt and decrypt messages which can be sent over the same secure connection. At no point did the symmetric key get communicated in *plain text*.

Revocation

Misissued or forged certificates, or certificates whose private key might have been compromised, can be revoked or held back during investigation of suspicious activity. A certificate may also need to be revoked for more innocuous reasons, such as the change of some identifier associated with the certificate. An expired certificate does not need to be revoked. Two standards for revocations are in common use:

| Name | Standard |
|--|---------------|
| CRL (<i>Certificate Revocation List</i>) | RFC 5280 [60] |
| OCSP (<i>Online Certificate Status Protocol</i>) | RFC 6960 [63] |

A **CRL** provides a publicly accessible repository of revoked certificates, identified by their serial number, with the timestamp of revocation attached. Each CA accepts CRL requests by entities they issued certificates for. Each CA can then attach the certificate in question to their CRL, and then regularly publish updates of their CRL. In order to determine whether a certificate is valid, a client will verify the signature using the referenced CA's certificate, check that the root certificate is found in its trust store, check the validity periods, *and* make sure the certificates are not on the CRL corresponding to the CA.

Updates to CRLs may be published with a delay, but knowing the *current* status of a certificate is sometimes critical. Another problem is the potentially large size of each CRL, and the burden put on the client to store these lists and read from them [64], [65]. In 2022, browser vendors introduced some improvements into CRL lookups, utilizing browser-specific summaries of revoked certificates issued by several CAs, as well as probabilistic algorithms, minimizing the size of CRLs [66].

The **OCSP** provides an alternative approach which allows the client to make a request to the issuing CA, which in turn requests a dedicated *OCSP responder* about a specific set of certificates. The CA then responds with each certificate's status.

The advantage is that a CA is now leveraged to check a certificate's status, thus improving performance. The *disadvantage* is that *a CA is now leveraged to check a certificate's status*, which is a privacy concern. A general problem is that a certificate's status cannot be retrieved if the OCSP service is not available and a client's CRL is outdated, but the CRL server is *also* down. Web browsers

have been opting to simply ignore the response if it does not come back quickly enough, which is a risk, but the alternative would be unavailability or severe latency of many websites [64].

OCSP Stapling is another approach in which a current, time-limited OCSP response, signed by the CA, is sent together with the TLS certificate by the TLS server itself [67]. Now, the browser need not consult the CA itself, increasing performance even further *and* solving the privacy issue. In order to force such an OCSP response, the browser needs to include a special flag, *Must-Staple*, in its request. However, if OCSP stapling goes wrong, the TLS server is not in control, so it cannot resolve any issues.

A remaining problem is the threat of a compromised CA, CRL server, or OCSP responder. *Certificate Transparency* (CT), mentioned in section 2.2, is one methodology that can be used additionally to make sure that revocations—and any other part of a certificate’s or CA’s life cycle, for that matter—are being monitored for suspicious activity [64]. *CT helps* in identifying certificates that possibly need to be revoked, but certificate revocation itself remains a hard problem.

3.2 The Domain Name System (DNS)

The purpose and architecture of the DNS is documented and specified in RFC 1034 [68] and RFC 1035 [69]; some important terminology is defined in RFC 2181 [70] and RFC 7719 [71]. Some important aspects of the history and function of the DNS have already been explained in the introduction, section 1, and the background, section 2.1. This section focuses on a few concepts necessary to explain how DANE works: Resource Record structure, DNS names, DNS zones, and name servers.

A Resource Record (RR) contains information associated with a specific name. The general structure of an RR includes:

| Part | Explanation |
|--------------------|---|
| Name | Refers to a DNS name, i.e. a hostname or “subdomain name” |
| Time To Live (TTL) | Tells a DNS cache how long the record should be cached for, in seconds, before expiring |
| Class | Used for rare or historical purposes; here, this will always be <i>IN</i> , short for <i>Internet</i> |
| Type | The type of record, explaining the usage of the data |
| Data | The actual data of the record |

All RRs that have different *data*, but have the same *name*, *TTL*, *class*, and *type* form a Resource Record Set (RRSet).

Names are relative to a *DNS zone*, which is an administrative portion of the DNS, identified by a *fully qualified name*. Zones are structured hierarchically and are all descendants of the *root zone* [72]. For example: `example.com.` is the name of a DNS zone, which is a child zone of the `com.`

zone, which is a child zone of the `.` zone (i.e., the root zone). A DNS entry of this zone might look like this:

```
www                18952   IN   A                93.184.216.34
```

`www` is the name, 18952 is the TTL (5 hours, 15 minutes, 52 seconds), `IN` is the class, `A` is the type (meaning that this record identifies the IPv4 address), `93.184.216.34` is the data (identifying the IPv4 address). If this RR sits in the `example.com.` DNS zone, then a DNS lookup for the name `www.example.com.` and type `A` will tell a client that the IPv4 address of that name is `93.184.216.34`. A special name is `@`, which refers to the name of the DNS zone itself, called the “zone apex” or “naked domain”.

However, the relative DNS name `www` requires the context of the DNS zone, identified by a RR of type `SOA` (*Start Of Authority*). To represent an individual record unambiguously, its name needs to be expanded with the name of the DNS zone, creating another *fully qualified name*, in this case `www.example.com.`. Therefore this example is a more common sight:

```
www.example.com.   18952   IN   A                93.184.216.34
```

These rows are in the so-called *presentation format*, a human-readable representation of RRs. Over the wire, the *wire format* is used instead, a compact binary format. The TTL is represented as a 4-byte number. The class and type are converted to 2-byte numbers, assigned by the relevant protocol standards—e.g. 1 for the class `IN`; 1 for the type `A`, 15 for the type `MX` (identifying the name of a mail server), etc. The data is split into a 2-byte length descriptor, the “`RDLENGTH`”, and an “`RDATA`” field, whose length is specified by the “`RDLENGTH`”, in bytes. The expression of the name over the wire is slightly more complicated: a sequence of lengths and individual labels (e.g. 3 `www` 7 `example` 3 `com` 0).

In the model presented in figure 1.1, the RR and RRSet type we care most about is the `TLSA` type for DANE. The following relationships so far are visualized:

- One *DNS zone* has authority over one or more *DNS names*
- One *DNS name* identifies zero or more individual *TLSA RRs*
- One *TLSA RRs* belongs to one specific *TLSA RRSet*
- One *TLSA RRSet* contains one or more *TLSA RRs*
- Any one *TLSA RR* is identified by one specific *DNS name*
- Any one *DNS name* authoritatively belongs to one specific *DNS zone*
- One *DNS zone* can contain zero or more *TLSA RRSet*

For the sake of simplicity, the *class* and the *TTL* of all `TLSA` RRs will be the same, so they do not play a major role in identifying different RRSets. Therefore:

- One *DNS name* contains at most one *TLSA RRSet*

In order to understand the security aspect of the DNS a bit better, it is worth briefly taking a look at the purpose of *name servers*. Name servers are servers that have two main functions:

- For requests about the zone they are *responsible* for—i.e. if the name server is an authority of the zone by virtue of being listed as one of the NS RRs within that zone—, they directly serve the DNS RRs of the zone as an *authoritative answer*;
- For other requests, they respond with a *non-authoritative answer* by either
 - performing a DNS lookup themselves (or looking into their cache), then caching the response with the RRs they receive, and then sending it back to the original requesting entity, or
 - sending a response back to the original requesting entity, telling them which other name server to request, which is likely to have an answer.

Whenever a client performs a DNS lookup, it will eventually receive an answer, either directly or indirectly.

An interesting question is how these name servers receive updates, especially regarding security considerations, and how name servers accept the authority of each other. If a DNS administrator chooses to update the RRs in their zone, they will perform a *zone transfer*. Best practice suggests, that a DNS zone has at least *two* name servers [73], which are only *secondary* servers, i.e. serving only a read-only copy of the zone data. Another name server, the *primary* server, should exist, which contains the *real* zone data, pushes updates to the secondaries, and is not publicly reachable in the Internet—only the secondaries have access to it [74]. This concept is known as *hidden primary*. Making the secondary servers only accept zone transfers from the hidden primary, keeping credentials for server access safe, keeping the secondaries physically separate, and ensuring the physical security of all servers are all important factors when considering name server security [75], [76]. Additionally, other security protocols exist, covered in section 3.5.

3.3 DNS Security Extensions (DNSSEC)

DNSSEC is specified in RFC 4033 [2], RFC 4034 [77], and RFC 4035 [78]. The reason why DNSSEC has been established as an extension to the DNS, is explained in the background, section 2.1. The *Internet Assigned Numbers Authority* (IANA) has published a set of general DNS and DNSSEC guidelines [73]. RFC 1912 also explains general tips for correctly configuring DNS [79].

This section explains how DNSSEC works. As previously stated, the DNS Security Extensions provide a public-key infrastructure (PKI) for the DNS in order to achieve data integrity and data origin authentication. This means it involves public and private keys, as well as signatures created from the private key. A DNSSEC-validated DNS response ensures the authenticity of the response data—this means a client can be confident that it will arrive at the desired destination.

DNSSEC introduces a few new record types that every zone can use, the important ones being listed in the following table:

| Type | Meaning |
|--------|-------------------|
| RRSIG | RRSet Signature |
| DNSKEY | Public key |
| DS | Delegation Signer |
| NSEC | Next Secure |

In short, the public key is stored in a DNSKEY RR; the corresponding private key signs an entire RRSet, which is stored in an RRSIG RR. The RRSIG has a validity period, so the zone needs to be periodically re-signed. Since an RRSIG RR cannot be easily and securely revoked, it is recommended to not make the validity period too long, especially for critical Resource Records [80].

Every zone has *two* underlying types of public and private key pairs:

| Key type | Public key | Private key |
|---------------------------------|-------------------------------------|--|
| ZSK (<i>Zone-signing key</i>) | Stored in a RR of type DNSKEY | Used to produce the signature of any RRSet, which is stored in an RR of type RRSIG |
| KSK (<i>Key-signing key</i>) | Stored in another RR of type DNSKEY | Used to produce the signature of the DNSKEY RRSet, specifically, which is stored in another RR of type RRSIG |

A child zone (such as `example.com.`) *owns* its DNSKEY RR with the name `example.com.`. The *parent* zone (`com.`) *owns* a RR of type DS and the name `example.com.`, which includes a digest of the public KSK from the child's DNSKEY. The name of both records is the same, but the ownership is not [81]. It is the responsibility of the DNS administrator of the `example.com.` zone to transmit the digest of their public key to the administrator of the parent zone, so that it can be inserted there. The parent zone then *vouches* for the child zone by virtue of a valid DS RR. The parent zone has its own DNSKEY RR, vouched for by *its* parent zone's DS RR, and so on, until eventually reaching the root zone, whose public key digest is publicly known. These links form the *chain of trust* throughout the hierarchy of a DNS zone, from a leaf zone up to the root zone. This is visualized in figure 3.1. Together with the signing of RRSets, this defines the public-key infrastructure of DNSSEC.

Finally, NSEC can tell a client what the *next* child zone in alphabetical order is. This record type exists for the purpose of authenticating the *denial of existence* of a specific DNS name. It prevents an attacker from forging a DNS response that contains RRs while the real response contains none; it also prevents an attacker from doing the opposite: forging a DNS response that contains *no* RRs while the real response contains *some*.

A client or a DNS server can always validate DNS responses using DNSSEC, though different types of DNS resolvers exist, not all of which verify DNSSEC signatures. The validation result ranges from *secure* (all keys, digests, and signatures correctly match), over *bogus* (some keys, digests, or signatures don't match, or some RRSIG RRs are expired), to *insecure* or *indeterminate* (no DNSSEC deployed). If the response is not deemed to be fully *secure*, e.g. due to being tampered with, the

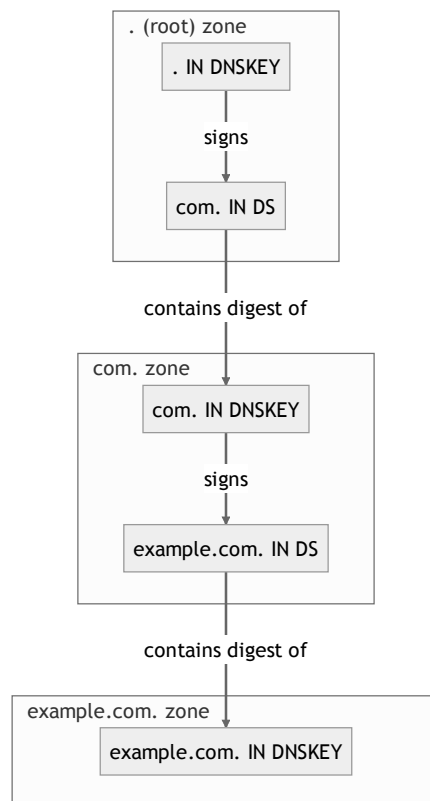


Figure 3.1: Chain of trust in DNSSEC

data simply will not reach its destination; the response will effectively tell the client that no data was retrieved [82].

In the context of *managing* DANE identities, particularly using TLSA RRs, we will assume that DNSSEC is fully deployed, and all relevant DNS zones are correctly signed and secure. This is because DANE is based on DNSSEC—or more accurately: DNSSEC is a *security requirement* for DANE. It is not a *technical* dependency, but DANE without DNSSEC is not useful—and the question is not really “*Why is DANE based on DNSSEC?*”. Instead, there are three facts to consider:

- DANE enables the authentication of servers using public key or certificate associations
- DANE is a set of *DNS-based* protocols
- The DNS protocol is inherently not very secure because it is susceptible to a variety of data manipulation attacks

If data is possible to be manipulated midway, no authenticity, no integrity can be guaranteed; any benefit, that employing DANE would have provided, is lost. It is akin to locking a door with a high-quality key and then hiding the key under the doormat, for any sufficiently diligent intruder to find. DANE is “based” on DNSSEC in the sense that DANE is a set of security protocols that are based on the DNS—and we need security throughout *all* protocol layers in order to benefit from security at any *one* layer.

3.4 The DANE TLS association (TLSA)

TLSA is the most interesting binding for DANE, because it relates to the TLS protocol which pervades everyday usage of the Internet. The first specification of TLSA has been published in August 2012 as RFC 6698, which proposes a protocol to allow DNS zone administrators to deposit the certificate data or public key of the TLS certificate of the corresponding TLS server into their DNS zone—either verbatim or in the form of a digest. The intended benefit of using the TLSA Resource Record is to avoid the dependency on Certificate Authorities to sign and issue certificates for websites [4]. Originally, an implementation of this protocol did not involve any particular change in the workings of a TLS server—only a TLS client would need to be extended. However, in October 2015, the protocol has been updated by RFC 7671, simplified in certain areas and extended in others. New guidelines and recommendations have been detailed, and new requirements for both TLS clients and TLS servers have been introduced [80]. The section will summarize the specifications in order to extract the guidelines and recommendations needed for an identity manager.

The purpose of the TLSA RRSet is to *authenticate* a TLS server. The benefit of *authentication* is explained by RFC 7671 as follows [80]:

Used without authentication, TLS provides protection only against eavesdropping through its use of encryption. With authentication, TLS also protects the transport against man-in-the-middle (MITM) attacks.

The general mechanism of authentication involves the client making a request to the TLS server as well as to the DNS. The TLS server responds with its certificate chain (part of the TLS handshake) while the DNS responds with the TLSA RRSet. The client then either compares the certificates or the public keys of both pieces of information directly—after hashing and extraction of the public key, if necessary—, or it validates the signature of the TLS certificate using the trust anchor certificate referenced in the TLSA RR. The client only needs *one* TLSA RR for the authentication, but if the client does not support any of the parameter combinations found, it will not use DANE, and, instead, proceed with the TLS handshake normally. If authentication fails, the client will abort the TLS handshake. This process is visualized in figure 3.2.¹

Opportunistic Security is a validation approach where a client will check if DNSSEC-validated, supported TLSA RRs exist, and if they do, then DANE authentication must succeed in order to establish a TLS connection. If such records do not exist, default to unauthenticated TLS. If no TLS is available, default to an unencrypted connection.

Correct DANE deployment hinges on correct DNSSEC deployment. If a TLSA RRSet is found for a particular domain, then a DANE client must use it if and only if the DNSSEC validation state is *secure*. If it is *bogus*, the TLS handshake must be aborted or not started to begin with. If it is *insecure* or *indeterminate* the TLSA RRSet shall not be used for a TLS connection. Successful DNSSEC validation serves the same purpose as the traditional PKI with CAs: trusted keys sign untrusted keys. RFC 6698 points out certain advantages of DNSSEC over Certificate Authorities: the public keys published into the DNS are directly tied to DNS names, distribution of keys over DNSSEC is more straightforward, and there is only a single parent zone that vouches for the child zone using signatures, instead of hundreds of possible CAs. RFC 6698 also points out [4]:

While the resulting system still has residual security vulnerabilities, it restricts the scope of assertions that can be made by any entity, consistent with the naming scope imposed by the DNS hierarchy. As a result, DANE embodies the security “principle of least privilege” that is lacking in the current public CA model.

The TLSA Resource Record

The data of a RR of type TLSA consists of four fields:

- The TLSA parameters, each an unsigned one-byte number
 1. Usage
 2. Selector
 3. Matching type
- A field of arbitrary length
 4. Certificate data

¹In the sequence diagram, the box with the “par” label and the two regions indicates two processes that execute *in parallel*. The box with the “break” label is a conditional execution depending on the condition written inside.

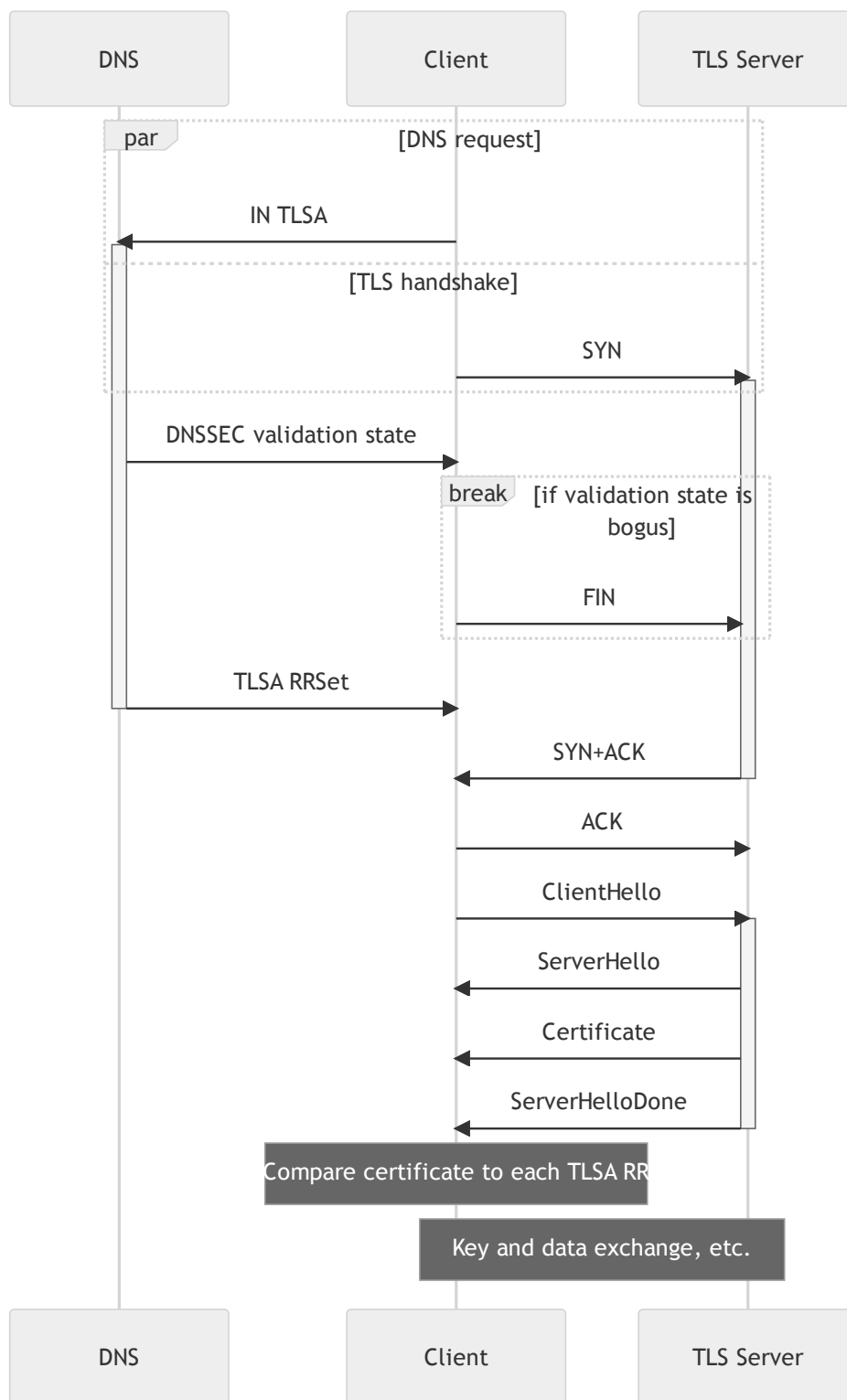


Figure 3.2: Example flow of TLS server authentication with a DANE-validating client

Each of the TLSA parameters has a specific set of numbers as possible values. The numbers also have IANA names associated with them, defined in RFC 7218 [83], and, separately, RFC 6698 [4] defines some labels for each usage.

The possible values for the **Usage** field are:

| Value | IANA name | RFC 6698 label |
|-------|-----------|--------------------------------|
| 0 | PKIX-TA | CA constraint |
| 1 | PKIX-EE | Service certificate constraint |
| 2 | DANE-TA | Trust anchor assertion |
| 3 | DANE-EE | Domain-issued certificate |

Usages are explained in more depth later in this section.

The possible values for the **Selector** field are:

| Value | IANA name |
|-------|-----------|
| 0 | Cert |
| 1 | SPKI |

“SPKI” refers to the *Subject Public Key Info* of a certificate, which includes the *modulus* and *exponent* of the public key, as well as the *algorithm*. These are technical details on how such a public key is stored, but the details are not important for defining a TLSA RR.

The possible values for the **Matching type** field are:

| Value | IANA name |
|-------|-----------|
| 0 | Full |
| 1 | SHA2-256 |
| 2 | SHA2-512 |

Given the possible values, there are $4 \times 2 \times 3 = 24$ possible *parameter combinations* for a TLSA RR. Different kinds of TLSA RRs are usually referred to by their parameter combinations in the order *usage*, *selector*, *matching type*. For example, one might say that a certain domain stores a certificate in a “2 0 0 TLSA record” or utilizes a “3 1 1 TLSA record”.

Finally, the **Certificate data** consists of raw bytes in the wire format, but is represented as a hex string in presentation format. The **Selector** and **Matching type** fields determine the content of the **Certificate data**. Basically, the **Selector** tells you if the TLSA RR involves the full TLS certificate or just its public key, and the **Matching type** tells you which algorithm, if any, has been used to hash this particular data, whose digest is used as the **Certificate data**.

The TLSA RR is located in the DNS under a name constructed from the port, the transport protocol, and the DNS name providing the target service (e.g. `www.example.com`, providing the Web service by having an IP address of a server that serves content via HTTP). For HTTPS, this DNS name would be `_443._tcp.www.example.com`, because HTTPS is served over TCP and port 443 by default. For SMTP, given a DNS name for a mail server such as `mail.example.com`, the resulting DNS name would be something like `_25._tcp.mail.example.com`. The “underscored” labels like `_443` and `_tcp` are subject to the scoped interpretation rules of RFC 8552 [84]; DNS names must not have *arbitrary* labels that start with an underscore (`_`).

In the model at figure 1.1, the final links can now be explained:

- One *TLS certificate* can be involved in zero or more *TLSA RRs*.
- One *Public key* can be involved in zero or more *TLSA RRs*.
- One *TLSA RR* corresponds to either exactly one *Public key* or exactly one *TLS certificate*, which includes exactly one *Public key*.

The final relationship appears to contradict the definition of an RRSset. However, it is just an *abstraction*:

- One *TLSA RRSset* is identified by one **or more** *DNS names*

This abstraction can be implemented using the *indirection* provided by the CNAME RR (*canonical name*). Consider these example RRs:

| | | | | |
|-------------------------------|------|----|-------|-------------------------------|
| <code>web.example.com.</code> | 3600 | IN | CNAME | <code>www.example.com.</code> |
| <code>m.example.com.</code> | 3600 | IN | CNAME | <code>www.example.com.</code> |
| <code>www.example.com.</code> | 3600 | IN | A | <code>93.184.216.34</code> |

The two CNAME RRs tell a DNS resolver to treat both the name `web.example.com.` and the name `m.example.com.` as if it is the name `www.example.com.`. As a result, `web.example.com.`, `m.example.com.`, and `www.example.com.` will all have the same IPv4 address, `93.184.216.34`. This is quite useful in simplifying DNS management.

Something similar can be achieved with TLSA RRs, therefore, backtracking every indirection, it can be said that one TLSA RRSset can be “reused” by multiple DNS names. This is what TLSA used with CNAME can look like:

| | | | | |
|---------------------------------------|------|----|-------|---|
| <code>a.example.com.</code> | 3600 | IN | A | <code>192.0.2.1</code> |
| <code>b.example.com.</code> | 3600 | IN | A | <code>192.0.2.2</code> |
| <code>_443._tcp.a.example.com.</code> | 3600 | IN | CNAME | <code>tlsa._dane.example.com.</code> |
| <code>_443._tcp.b.example.com.</code> | 3600 | IN | CNAME | <code>tlsa._dane.example.com.</code> |
| <code>tlsa._dane.example.com.</code> | 3600 | IN | TLSA | <code>2 0 1 (E3B0C44298FC1C149AFBF4 C8996FB92427AE41E4649B 934CA495991B7852B855)</code> |

Now, `a.example.com` and `b.example.com` can both use the same DANE record. The aliasing to `tlsa._dane....` is directly recommended by RFC 7671 [80], and `_dane` is guaranteed to be an available underscored name by RFC 8552 [84].

Generating a TLSA RR

Given a TLS certificate *cert*, a desired **Usage** *u*, a desired **Selector** *s*, and a desired **Matching type** *m*, a TLSA RR, in presentation format, can be generated by following these steps:

1. Let *result* be the list « *u*, *s*, *m* ».
2. If *s* is 0 (Cert),
 1. let *selection* be the DER-encoded binary data of *cert*.
3. Else, if *s* is 1 (SPKI),
 1. let *selection* be the DER-encoded binary data of the *Subject Public Key Info* of *cert*.
4. If *m* is not 0,
 1. let *algorithm* be the hashing algorithm corresponding to the **Matching type** as chosen by *m*, and
 2. let *data* be `HexBytes(algorithm(selection))`.
5. Else, if *m* is 0,
 1. let *data* be `HexBytes(selection)`.
6. Append *data* to the end of *result*.
7. Return *result*, joined by a space (U+0020).

`HexBytes` is a helper algorithm that takes arbitrary binary data as input and returns a string by replacing every byte by its hexadecimal representation, padded with 0 from the left, with a length of 2 per byte.

Given the following example certificate, we apply this algorithm with the desired parameter combination of *u* = 3, *s* = 1, *m* = 1; *cert* is defined by the following certificate in PEM format:

```
-----BEGIN CERTIFICATE-----
MIIFHzCCBAegAwIBAgISA2Ve5XLN8+Ifr+UQjVKP91PiMA0GCSqGSIb3DQEBCwUA
MDIx CzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQYDVQQD
EwJSMzAeFw0yMzAxMDE4NDVhFw0yMzA0MDE4NDRaMBGx FjAUBgNVBAMT
DXd3dy5odXF1ZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC
RbopbZp5S3UaVKTkxze3J7MGzxh/i3pYupG2eE0M+h4op+esreqTEiQPDuzM22D
NzvMe8XLd+uyEPloTuIknNvGu/oqiQ0860PZBpiiZ7UUpCnPyj/8eLQzc3kyL6Do
TRFTYe+2sQ1X97KIxyQRuVPRtT1wRbbeqMSLc5KK8YP8XBxzAwfVEXw3anHqGzg4
6f2Hmp1C3s+5VVycy/22+1lRsse2ywWunJ0TVP6X82JR4n6VxPnPnsNyRMCSi1ob
KsGAVPjCvut6mU1iW75Azbdl1G8KhZ0nCJ0zsUzmz3Um09JQBNM0vw+D5srtgyc
```

```
VaiBygz9FoA9qe1TSvF/AgMBAAGjggJHMIICQzA0BgNVHQ8BAf8EBAMCBaAwHQYD
VR0lBBYwFAYIKwYBBQUHAWEGCCsGAQUFBwMCMawGA1UdEwEB/wQCMAAwHQYDVR00
BBYEFFaET0SUTRwq0+eFPPZYrrNL5QnHMB8GA1UdIwQYMBaAFBQusxe3WfbLrLAJ
QOYfr52LFMLGMFUGCCsGAQUFBwEBBEkwRzAhBggrBgEFBQcwAYYVaHR0cDovL3Iz
Lm8ubGVuY3Iub3JnMCIGCCsGAQUFBzACHhZodHRwOi8vcjMuaS5sZW5jci5vcmcv
MBgGA1UdEQQRMA+CDXd3dy5odXF1ZS5jb20wTAYDVR0gBEUwQzAIBgZngQwBAGew
NwYlKwYBBAGC3xMBAQEwKDAmBggrBgEFBQcCARYaaHR0cDovL2Nwcy5sZXRzZW5j
cnlwdC5vcmcwggEDBgorBgEEAdZ5AgQCBiH0BIHxA08AdgC3Pvsk35xNunXy0cW6
WPRsXfxCz3qfNcSeHQBjJe20mQAAAYVtRSRtAAAEAwBHMEUCIQDmUhmAFtFb2zly
5pl+6M9h904mKw00Q/3D7xLeC4UEXQIQHDHvHK1GXG026pX9tTAWDm5FBiu/j5L
x+hyhR4n/CwAdQDoPtDaPvUGNTLnVyi8iWvJA9PL0RFR70tp4Xd9bQa9bgAAAYVt
RSZeAAAEAwBGMEQCIDL3+ELYFK7YjSfM90/2fB65zSRMja8MH+8s2Eul/KtBaIBV
UFRBFXavwzDyvf7bx8Ac5HU2I9wmwV3gq47UukGNHDANBgkqhkiG9w0BAQsFAAOc
AQEAc5n5XsUA4o52KFGA0wVvG0KT98Fxy2qx9g2VvNuZ8sUy8k77gVJQyCFOiNEh
zh/sF9rs/qqEltrOfkzzh9503uEMTQLQf99i8pgzL+lvAd4MDAK6ZkfUenJWvMRQ
dojdB+XdCN8MxdWM36IhAXf1uXE1sBpgBrgclZ4Rx1n8v6pBExznSDVppQvptid
pIp8dXKxXHPsIu1XLB2wUJ9x0uVpSrXjKA3A5sDQxZYPmcBxZrtEt1EN0jQn/AxY
U50085HG7LMLrnBlldhiW0Ubxrphu3VgacUBpzJjg7lnln05aKLP/1sJ0bDCximFp
ugFlnz0t++E34vxYsd5tW6VPoA=
-----END CERTIFICATE-----
```

The DER-encoded binary data of the *Subject Public Key Info* of the above certificate can be represented by the following hex dump:

```
00000000  30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 |0.. "0 ... *.H.....|
00000010  01 05 00 03 82 01 0f 00 30 82 01 0a 02 82 01 01 |.....0.....|
00000020  00 a1 45 ba 29 6f 3a 79 4b 75 1a 54 a4 ca 6f 1c |..E.)o:yKu.T..o.|
00000030  de dc 9e cc 1b 3c 61 fe 2d e9 62 ea 46 d9 e1 34 |.....<a.-.b.F..4|
00000040  33 e8 78 a2 9f 9e b2 b7 aa 4c 48 90 3c 3b b3 33 |3.x.....LH.<;.3|
00000050  6d 83 37 3b cc 7b c5 cb 77 eb b2 10 f9 68 4e e2 |m.7;.{..w....hN.|
00000060  24 9c db c6 bb fa 2a 89 03 bc eb 43 d9 06 98 a2 |$.....*....C....|
00000070  67 b5 14 a4 29 cf ca 3f fc 78 b4 33 73 79 32 2f |g... ).. ?.x.3sy2/|
00000080  a0 e8 4d 11 53 61 ef b6 b1 0d 57 f7 b2 88 c7 24 |..M.Sa....W....$|
00000090  11 b9 53 d1 b5 3d 70 45 b6 de a8 c4 8b 73 92 8a |..S..=pE.....s..|
000000a0  f1 83 fc 5c 15 f3 03 07 d5 11 7c 37 6a 71 ea 1b |... \.....|7jq..|
000000b0  38 38 e9 fd 87 32 9d 42 de cf b9 55 5c 9c cb fd |88... 2.B... U\ ...|
000000c0  b6 fb 59 51 b2 c7 b6 cb 05 ae 9c 9d 13 54 fe 97 |..YQ.....T..|
000000d0  f3 62 51 e2 7e 95 c4 f9 cf 9e c3 72 44 c0 92 23 |.bQ.~.....rD..#|
000000e0  5a 1b 2a c1 80 54 f8 c2 be eb 7a 99 4d 62 5b be |Z.*..T....z.Mb[.|
000000f0  40 ce 26 dd 97 51 bc 2a 16 4e 9c 22 74 ce c5 33 |@.&..Q.*.N."t..3|
00000100  9b 3d d4 98 ef 49 40 13 4c 3a fc 3e 0f 9b 2b b6 |.=... I@.L.:>..+.|
00000110  0c 9c 55 a8 81 ca 0c fd 16 80 3d a9 ed 53 4a f1 |..U.....=..SJ.|
00000120  7f 02 03 01 00 01 |.....|
00000126
```

Finally, the SHA2-256 digest of the above looks like this in its hexadecimal form:

A67924AFCD895B9661C4C5D67A83215F60B7D0E1DA8A30B67EEC6BD623A5B57C

The final data of the resource record looks like this:

3 1 1 A67924AFCD895B9661C4C5D67A83215F60B7D0E1DA8A30B67EEC6BD623A5B57C

The exact process is explained in section 5.1.

The **Usage** does not play a role in generating the *content* of the **Certificate data**, but it very much plays a role in the client validation approach.

The standard allows the presentation format to include arbitrary whitespace and parentheses in the RR data. The above certificate happens to be issued for `www.huque.com`, therefore the full record may look like this:

```
_443._tcp.www.huque.com.      7200   IN   TLSA      3 1 1 ( A67924AFCD895B9661C4C5
                                           D67A83215F60B7D0E1DA8A
                                           30B67EEC6BD623A5B57C )
```

Usages and recommendations

The Usage field can be split into two bits:

| Bit | . 0 (“-TA”) | . 1 (“-EE”) |
|--------------|-------------|-------------|
| 0. (“PKIX-”) | PKIX-TA | PKIX-EE |
| 1. (“DANE-”) | DANE-TA | DANE-EE |

In a nutshell,

- “PKIX-” usages should be used for certificates issued or owned by a publicly known CA,
- “DANE-” usages should be used for self-issued or self-owned certificates (i.e. without a publicly known CA),
- “-TA” usages should be used for certificates that issue other certificates, and
- “-EE” usages should be used for leaf certificates.

More specific guidelines are summarized below.

A TLSA RR with the DANE-EE usage (3) must directly match an end-entity certificate (leaf certificate) or public key provided by the TLS server. A TLSA RR with the PKIX-EE usage (1) works similarly, but imposes additional requirements. The major difference between “PKIX-” and “DANE-” usages is that validation using “PKIX-” usages requires domain name checks, validity period checks from the certificate itself, as well as certification path checks against pre-installed CA certificates in a trust store. For the “DANE-” usages, no checks against pre-installed CA certificates in a trust

store are needed, allowing self-issued certificates to be used. For the DANE-EE usage, *all* of these checks are optional; the certificate can even be expired—although keeping expired certificates for too long is not recommended. The domain name checks refer to matching the domain names included in the certificate and the domain name a client uses to reach the server (the “reference identifier”); again, these names need not match with DANE-EE.² This is also why CNAME indirection works best for the “DANE-” usages. For “PKIX-” usages, indirection is a bit more complicated; this is explained in section 6 of RFC 7671 [80]. Furthermore, Certificate Transparency checks are applicable to the “PKIX-” usages, but not to the “DANE-” usages. In summary, DANE-EE is a more lenient, yet more fault resistant usage type than PKIX-EE. The benefit of “PKIX-” usages over not using TLSA RRs *at all* is that the TLSA RRs constrain the possible CAs involved in issuing end-entity certificates.

Server Name Indication (SNI) is a TLS extension allowing a client to identify a specific service on a server that offers multiple services on the same IP address. This extension is mandatory to be used and verified with all usages except DANE-EE. A DANE client is required to specify the “base domain” of the TLSA RRSet when using SNI—for example, for `_443._tcp.www.example.com`, the base domain is `www.example.com`. If name checks are required, then the TLS certificate must be issued for `www.example.com`. In an MX record, a mail server can be identified; the TLS certificate can *also* be issued for the mail server destination.

The other two parameters, Selector and Matching type, also have a few recommendations. For the hashing algorithm (matching type), SHA2-256 is mandatory for both client and server, whereas SHA2-512 is just recommended for the client—it is not to be exclusively provided in TLSA RRs. Publishing the full certificate or public key in the DNS is usually not recommended because the TLSA RRs would become very large, presenting problems with UDP transport.

Some interesting scenarios are enabled for both client and server. If a trust anchor certificate is published to the DNS within a TLSA RR, with no hashing performed, a client can retrieve the full trust anchor certificate from the DNS, so the TLS server need not necessarily send the full certificate chain. The client might extract the public key from the TLSA RR, and, with a certificate with “-TA” usages, it will perform certification path checks against it. However, sending the full certificate chain is still the recommendation for the TLS server in most cases, and DANE clients are not expected to fully support all of these scenarios. As for the selector, having only a public key association in the DNS, instead of the full certificate, allows a TLS server to negotiate raw public keys (RFC 7250 [87]) with a client; the TLS server, in theory, does not need to send the entire TLS certificate chain to the client. It also allows the server to keep a TLSA RR with the same public key while renewing the certificate. Raw public key negotiation is also possible with the full *certificate* in the DNS, but clients are not expected to support this scenario.

Briefly, a TLSA RR with the DANE-TA usage (2) can be used *in lieu of* one with the DANE-EE usage to publish a trust anchor certificate instead of an end-entity certificate to the DNS, while

²`openssl` always performs name checks by default without the `-dane_ee_no_namechecks` flag. This willful violation of the standard is due to “unknown key share” attacks and cross-origin scripting restrictions [85]. Unknown key-share (UKS) attacks exploit the lack of name checks. If only the public key is available in a TLSA RR, a DANE client will not perform proper identity proofing of the TLS server during the TLS handshake, tricking the client into believing the server has a different name, possibly circumventing firewall restrictions or same-origin restrictions [86].

the TLS server can serve one of (potentially) many end-entity certificates—and change them as frequently as desired—, without the need to publish them in the DNS. A client is expected to use this certificate association to validate the end-entity certificate from the server against the trust anchor certificate from the DNS, forming a complete certificate chain. A TLS server is expected to still serve the trust anchor certificate in its certificate chain, unless 2 0 0 TLSA RRs are used *exclusively* (i.e. the full, unhashed certificate). Again, self-issued certificates are encouraged with this usage, however, in this scenario, name checks must be performed against the end-entity certificate. If a certificate has a corresponding TLSA RR with the DANE-TA usage (2) and this certificate is revoked, the corresponding TLSA RR must be removed as soon as possible.

Finally, a TLSA RR with the PKIX-TA usage (0) can be used to specify a specific root or intermediate trust anchor certificate of a publicly known CA. It is possible that the specified trust anchor certificate is not found in the client’s trust store; a DANE client is expected to extend the certificate chain from the TLS server by considering the certificates found in the TLSA RR.

These are the recommended parameter combinations, based on RFC 7671 [80]. For the Matching type field, * is either 1 or 2.

| Usage | Selector | Matching type | Recommendation |
|-------|----------|---------------|--|
| 0 | 0 | 0 | Avoid: <i>record size too big</i> |
| 0 | 0 | * | Likely good ³ |
| 0 | 1 | 0 | Maybe: <i>compatible with raw public keys, size might be too big</i> |
| 0 | 1 | * | Likely avoid ³ |
| 1 | 0 | 0 | Avoid: <i>record size too big</i> |
| 1 | 0 | * | Likely good ³ |
| 1 | 1 | 0 | Maybe: <i>compatible with raw public keys, size might be too big</i> |
| 1 | 1 | * | Good ⁴ |
| 2 | 0 | 0 | Avoid: <i>record size too big</i> |
| 2 | 0 | * | Good |
| 2 | 1 | 0 | Avoid: <i>clients might not be able to validate this</i> |
| 2 | 1 | * | Avoid: <i>constraints from certificate are important</i> |
| 3 | 0 | 0 | Avoid: <i>record size too big</i> |
| 3 | 0 | * | Possibly good ⁵ |
| 3 | 1 | 0 | Maybe: <i>compatible with raw public keys, size might be too big</i> |
| 3 | 1 | * | Good |

³“0 0 *”, “0 1 *”, “1 0 *” do not have explicit recommendations by either RFC 6698 or RFC 7671, so their recommendation is based on similar parameter combinations.

⁴“1 1 1” is used as a common example, therefore we can assume that “1 1 *” is a recommendation for the usage PKIX-EE.

⁵The IETF DANE Working Group has recommended against “3 0 *” on the basis of a lack of reliable automation. The short-livedness of *Let’s Encrypt* end-entity certificates (90 days) requires a frequent change of the TLSA RR [88].

In general,

- a Matching type of 1 is required, but 2 can also be used *additionally*;
- storing the full, unhashed certificate in a TLSA RR should be avoided due to the large size and associated problems with UDP packages;
- raw public keys can be negotiated, but avoid them if the other constraints found in the certificate are important, or if the TLS server needs to be forced to transmit the full certificate during the TLS handshake.

Client and server requirements

The TLSA RRSSet can be grouped by parameter combination. For example, all TLSA RRs of a TLSA RRSSet with the parameter combination 3 1 1 form the “3 1 1 group”. For *every* such non-empty parameter combination group, the TLSA RRs within the group must be sufficient to authenticate a server. This means, for each such non-empty group, there must be at least one TLSA RR corresponding to the *current* certificate from the TLS server. It is acceptable—and in transitional situations (e.g. key rollovers), *required*—to keep TLSA RRs of *future* or *past* certificates. The SHA2-512 algorithm is not universally supported, so as a general rule, for every TLSA RR with Matching type 2, there must exist a TLSA RR with the same parameter combination, except with the Matching type 1.

The following list is a collection of other client and server requirements provided in RFC 7671. In this context, * means *any value for the given field*.

- For “0 * *”, “1 * *”, and “2 * *”, the validity period of the TLS certificate must *not* be ignored for validation
- For “2 * *”, if *not* all TLSA RRs have “2 0 0”, then the full TLS certificate chain is required from the server
- For “2 * 1”, and “2 * 2”, the TLS server must include the TA certificate in the certificate chain
- For “2 1 0”, the TLS server *should* include the TA certificate in the certificate chain
- For “3 * *”, the validity period of the TLS certificate must be ignored for validation
- For “3 1 *”, and, optionally, “3 0 0” the TLS server need not necessarily send the TLS certificate

Parameter combinations “0 * *” and “1 * *” work similarly, but with additional PKIX-related requirements, explained earlier in this subsection. Not having any TLSA RRs is, of course, allowed, but such a domain would not be supporting DANE (for TLS), by definition. Publishing both “PKIX-” and “DANE-” usages is allowed. It is currently recommended that a DANE client validates TLSA RRs of usages 2 (DANE-TA) and 3 (DANE-EE), but offers no support for 0 (PKIX-TA) and 1 (PKIX-EE) [80]. There are some additional requirements that are not directly related to TLSA RR management, e.g. the requirement to use at least TLS version 1.0 and the recommendation to use TLS version 1.2 for clients.

Because name servers cache RRs for some amount of time, the deployment of DANE-related RRs needs to consider the TTL of each record. In particular, the general procedure of publishing a new certificate involves publishing the corresponding TLSA record *first*, then waiting some amount of time to ensure old TLSA records have expired from any caches, and *then* publishing the certificate on the TLS side. The following is a summary of the TLSA *publisher* requirements from RFC 7671, Section 8.4 [80]—this is what this thesis is trying to establish.

In summary, server operators updating TLSA records should make one change at a time. The individual safe changes are as follows:

- *Pre-publish new certificate associations that employ the same TLSA parameters [...] as existing TLSA records, but match certificate chains that will be deployed in the near future.*
- *Wait for stale TLSA RRSets to expire from DNS caches before configuring servers to use the new certificate chain.*
- *Remove TLSA records matching any certificate chains that are no longer deployed.*
- *Publish TLSA RRSets in which all parameter combinations [...] present in the RRset match the same set of current and planned certificate chains.*

A TLSA RR can simply be removed from the DNS. Removing it will cause it to be eventually removed from all caches; at the latest, the validity period of the corresponding RRSIG RR will eventually expire. In order to mitigate key compromise, it is recommended to keep the TTL small, so a revoked and removed TLSA RR will be removed from caches sooner [80].

In section 4.2, various ways to *change* a TLSA RRSet will be examined. In section 5, the implementation aspects of these rules and the RRSet changes will be examined.

Other DANE bindings: IPSECKEY, SSHFP, OPENPGPKEY, SMIMEA

DANE is not “one thing”. There are currently five different “bindings” for DANE, i.e. five different certificate or public key associations. They are listed here:

| RR type | Standards | Usage |
|------------|------------------------------|--------------------------------|
| IPSECKEY | RFC 4025 | IPsec keying material |
| SSHFP | RFC 4255 | SSH fingerprint |
| TLSA | RFC 6698, RFC 7671, RFC 7672 | TLS certificate or public key |
| OPENPGPKEY | RFC 7929 | OpenPGP public key |
| SMIMEA | RFC 8162 | S/MIME certificate association |

3.5 Other DNS- and TLS-related security protocols

There are several other protocols related to Internet security that might need to be considered in coordination with DANE. This section provides an overview and briefly describes them—these three types of questions will be addressed:

- *How is security aspect X achieved?*
- *How is part Y of the Internet secured?*
- *Where exactly does DANE fit into the realm of DNS security?*

This is an overview of relevant protocols and their specifications:

| Protocol | Specification |
|--|----------------|
| DoT (<i>DNS over TCP</i>) | RFC 7766 [89] |
| DoH (<i>DNS over HTTPS</i>) | RFC 8484 [90] |
| TSIG (<i>Transaction Signature</i>) | RFC 2845 [91] |
| DNSCurve | RFC draft [92] |
| TLSA for SMTP (<i>Simple Mail Transfer Protocol</i>) | RFC 7672 [93] |
| TLS server identity check for email-related protocols | RFC 7817 [94] |
| DMARC (<i>Domain-based Message Authentication, Reporting, & Conformance</i>) | RFC 7489 [95] |
| HSTS (<i>HTTP Strict Transport Security</i>) | RFC 6797 [96] |
| HPKP (<i>Public Key Pinning Extension for HTTP</i>) | RFC 7469 [97] |
| CAA RRs (<i>Certification Authority Authorization</i>) | RFC 6844 [98] |
| HTTPS and SVCB RRs | RFC draft [99] |
| ACME (<i>Automatic Certificate Management Environment</i>) | RFC 8555 [100] |

The purposes of these protocols are as follows:

- **DoT** specifies the usage of TCP as a transport protocol for DNS, as opposed to UDP. It allows larger message sizes (needed for DNSSEC) and certain security features.
- **DoH**, additionally, specifies DNS transport over HTTPS (and thereby also TLS), i.e. *encrypted*—this is what provides *confidentiality* and additional *integrity* in DNS transport.
- **TSIG** is used for authenticating *DNS transactions*, e.g. zone transfers. It involves timestamps to prevent replay attacks, i.e. intercepting and re-sending a packet—even if encrypted. However, it can only secure the immediate connection between two name servers; it does not provide end-to-end integrity like DNSSEC [5].
- **DNSCurve** provides encryption and authentication of DNS packets between name servers. It attempts to provide *confidentiality*, *integrity*, and even some *availability* by defending against replay attacks, discarding forged DNS packets.
- **TLSA for SMTP** uses DANE TLSA RRs for mail server authentication during e-mail transport using the SMTP.
- **RFC 7817** defines rules for a “*consistent TLS server identity verification procedure across multiple email-related protocols*”, which can be used in tandem with DANE.

- **DMARC** is a mechanism that allows mail servers to “*associate reliable and authenticated domain identifiers with messages, communicate policies about messages that use those identifiers, and report about mail using those identifiers*”.
- **HSTS** provides a policy for websites or web browsers to force a secure connection over HTTPS (and TLS) by default.
- **HPKP** is a deprecated protocol which was used to provide a policy for websites to force a specific certificate or public key to be used in a TLS connection.
- The **CAA RR** provides a policy for authorizing only specific CAs for a particular domain, reducing the risk of misissuing.
- The **HTTPS RR** and **SVCB RR** provides a mechanism to optimize connection negotiation between a client and a server via the DNS, and promises to improve performance and privacy.
- **ACME** offers automatic authentication of a domain for issuing a TLS certificate by receiving a token from the issuing CA and placing it either into the DNS zone (“DNS-01 challenge”) or the web service (“HTTP-01 challenge”). The CA will then attempt to request a TXT RR (used for arbitrary text) from a specific underscored domain or make an HTTP request to a specific end point, and check if the token is correct.

Figure 3.3 is a rough visualization of where these protocols play a role.

DNSSEC and DANE are only part of the puzzle. In order to maximize security, several protocols need to be considered together, as well as access control and physical server security. But, as mentioned before, it is required to weigh the cost and benefit of every protocol. Naively accumulating several individual protocols that promise to increase security, without proper risk assessment, might in turn impair performance or privacy, ironically *decreasing* security (especially availability).

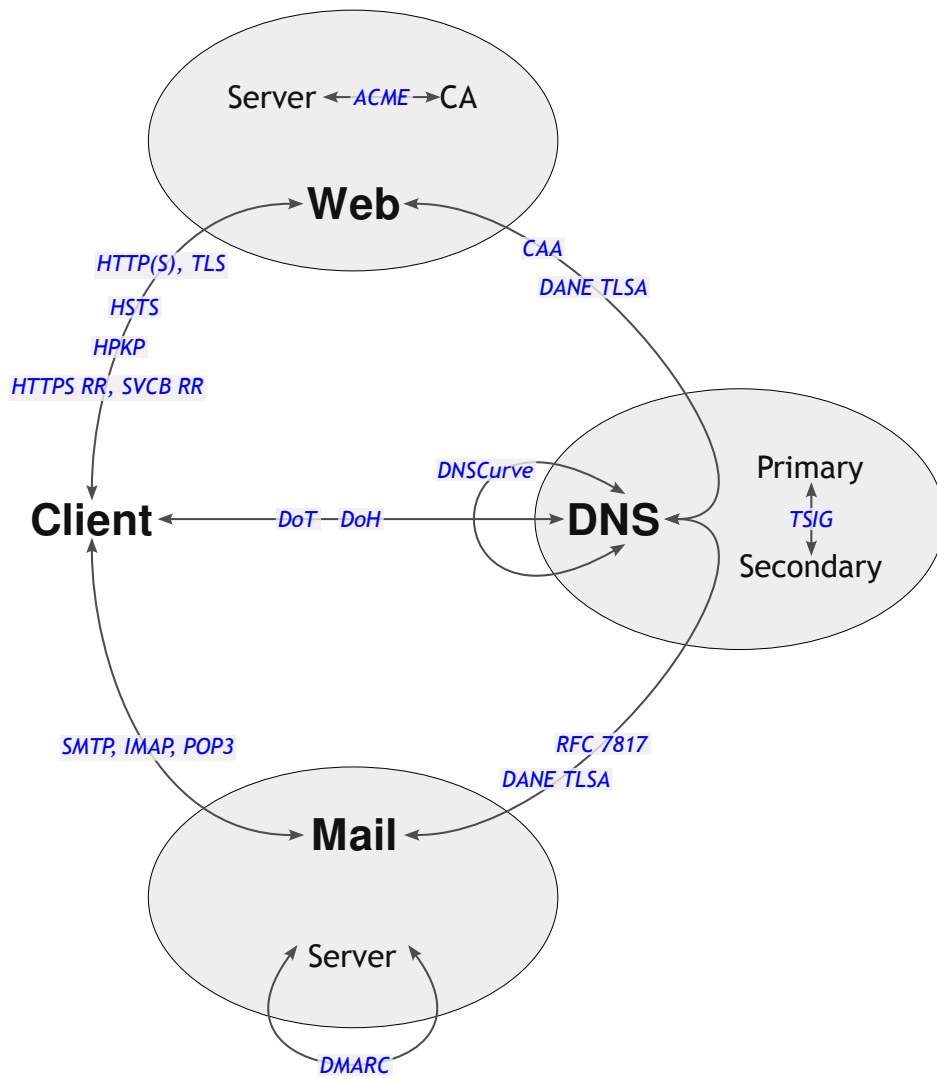


Figure 3.3: Protocols involved in DNS, Web, and Mail

4 The standards aspect of DANE identity management

This section examines various standards documents that are about how to approach identity management in general, not about any specific protocol or implementation. First, we clarify the terminology needed to understand the field of identity management. Next, we take a look at the identity life cycle and how it relates to DANE identity management.

4.1 Terminology

The title of this thesis implies that we are going to *manage identities*. Before we can do that, we need to understand what these so-called “identities” *are*. This will uncover the complex taxonomy involved in identity management.

Attribute, identifier, entity, identity

We can start with the term **identity**. The Internet Security Glossary (RFC 4949) offers the following definition [56]:

The collective aspect of a set of attribute values (i.e., a set of characteristics) by which a system user or other system entity is recognizable or known.

It further explains that the set of attributes used to define identities “*should be sufficient to distinguish each entity from all other entities, i.e., to represent each entity uniquely*” and that the set “*should be sufficient to distinguish each identity from any other identities of the same entity*”.

The ISO standard for “*A framework for identity management*” (ISO/IEC 24760-1) defines *identity* (or *partial identity*) as follows [101]:

Set of attributes related to an entity.
Note: An entity can have more than one identity.
Note: Several entities can have the same identity.

Whether one should be able to *uniquely* identify an entity depends on the context. Security vulnerabilities happen when a client is led to believe that a server has an identifier that, in reality, should

only identify *another* server—either accidentally or maliciously—; this is what impersonation, man-in-the-middle attacks, etc. are all about. The uniqueness aspect of the *identity* term, as well as how to approach the confusion between two identities, are both important to specify.

Entity and *attribute* are central terms that help define what an *identity* is. As for **entity** (or *system entity*), RFC 4949 defines it as [56]:

An active part of a system—a person, a set of persons (e.g., some kind of organization), an automated process, or a set of processes [...]—that has a specific set of capabilities.

And the ISO standard says it is the following [101]:

Item relevant for the purpose of operation of a domain that has recognizably distinct existence.

Note: An entity can have a physical or a logical embodiment.

Example: a person, an organization, a device, a group of such items, a human subscriber to a telecom service, a SIM card, a passport, a network interface card, a software application, a service, or a website

An **attribute** is simply defined by ISO [101] as this:

Characteristic or property of an entity

Example: An entity type, address information, telephone number, a privilege, a MAC address, a domain name are possible attributes.

And, finally, RFC 4949 [56] defines it as this:

Information of a particular type concerning an identifiable system entity or object

A special type of attribute is the **identifier**. RFC 4949 defines it as [56]:

A data object—often, a printable, non-blank character string—that definitively represents a specific identity of a system entity, distinguishing that identity from all others.

And ISO defines it as [101]:

Attribute or set of attributes that uniquely characterizes an identity in a domain

Another common theme of the usage of the word “*identifier*” is the fact that an identifier is an *external* description, meaning that the entity can be reached by it from an entity outside of the system that the target entity belongs to.

In the Wikipedia article about *Identity management*, the relationship between *entity*, *identity*, and *attribute* is explained by a so-called “axiomatic model”, shown here in figure 4.1 [102].

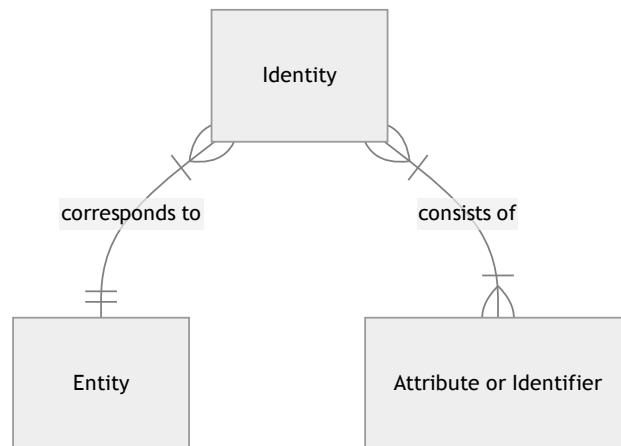


Figure 4.1: “Axiomatic model” of an identity

Essentially, an identity corresponds to an entity, i.e. a subject or object, and consists of a few attributes that describe the identity. The identifier has the purpose of unique **identification** of an identity, i.e. “*recognizing an entity in a particular domain as distinct from other entities*” [101].

RFC 6125 distinguishes two terms for identity (or identifier), based on *perspective*, as shown in the following table [103]. Note that both terms refer to the *same* identity: the identity of the server. Based on who is referencing this identity, the term—and, in fact, the attributes, in some cases—will differ.

| Identity in question | Entity referencing this identity | Term |
|------------------------|----------------------------------|---------------------------|
| Identity of the server | Server | Presented identity |
| Identity of the server | Client | Reference identity |

Figure 1.1 models the relationship between *Identity* and different attributes. For example, some *Public key* can be an attribute for some usage. The identifier is linked to one or more *DNS names*, transitively via one or more *TLS certificates*. An actual TLS certificate also includes the subject’s name, their email, their organization’s name, etc., all of which are attributes associated with the identity. The *entity* is not part of the diagram, but a *Website* or *Web service* can be considered to be the relevant entity. However, Web services are not in the scope of identity management that this thesis is about. Instead, the *TLS certificates* serve as a proxy for Web services in that diagram.

The ISO standard mandates that an identity represents an entity, and presumes that the identity is persistently stored [101]. There are two immediate ideas as to what the *entity* is that a DANE identity corresponds to: either a certificate or a public key. But TLS certificates only have a limited lifetime. Public keys can be persisted across several certificate renewals, yet it is also recommended

to rotate them once in a while. However, TLS certificates are also linked to one or more *names*, which is an important attribute of an identity. Names—or, more specifically, *hostnames*—are persisted over a significantly longer lifetime. The expected long lifetime of a hostname and the fact that a certificate applies to a specific set of hostnames, even after being renewed, suggest a third, arguably more intuitive, idea: the entity could be a “track” of TLS certificates which are issued for the same hostnames and being continually renewed. Only one certificate can be the *currently used* one; it will have a different serial number, a different validity period, maybe a different public key as the previous one—but it will belong to the same *track* as the previous one.

Ultimately, an identity is a collection of attributes, so, if we accept the third option as the definition of *entity*, we can deduce the relevant attributes required to describe the *identity*. The ISO standard proposes the following checklist for what kind of attributes need to be supplied [101]:

- *Any information required to facilitate the interaction between the domain and the entity for which the identity is created;*
- *Any information required for future identification of the entity, including description of aspects of the physical existence of the entity;*
- *Any information required for future authentication of the entity’s identity; or*
- *One or more reference identifiers.*

The following list provides a minimal set of attributes relevant for DANE identity management; an actual implementation might require more knowledge.

- The *current* certificate in use, from which one can derive:
 - the list of hostnames that the current certificate applies to, and
 - the validity period
- Information needed for renewal of the certificate, such as:
 - a stored certificate signing request (CSR), and
 - authentication tokens for CAs
- The deployment state of the current certificate
- The *current* TLSA RRs that correspond to the certificate, including:
 - their deployment states
- The policies to be applied when generating the TLSA RRs
- References to future or past certificates needed for preparation of TLSA RRs, logging, etc.
- Perhaps a human-readable, easily identifiable label, which can serve as the identifier

Authentication, Verification

The central operation one can perform with an identity is *authentication*. This is exactly what *authenticity* is: communication with some entity is authenticated if and only if a subject has managed to prove that the entity that performs some action, such as sending a message, truly is the entity that

the subject expects it to be; the subject is trying to *verify* the entity's identity. And this, of course, is exactly what explains the "Authentication" in "DNS-based Authentication of Named Entities".

The ISO standard defines "**authentication**" as [101]:

Formalized process of verification that, if successful, results in an authenticated identity for an entity

Note: The authentication process involves tests by a verifier of one or more identity attributes provided by an entity to determine, with the required level of assurance, their correctness

RFC 4949 concurs with its definition of "**authenticate**" [56]:

Verify (i.e., establish the truth of) an attribute value claimed by or for a system entity or system resource.

Further, under the definition of *authentication*, it points out, that authentication consists of two steps:

1. Identification – "*Presenting the claimed attribute value [...] to the authentication subsystem*"
2. Verification – "*Presenting or generating authentication information [...] that acts as evidence to prove the binding between the attribute and that for which it is claimed*"

In DANE, the *identification* is simple: the identifier is the hostname, e.g. `www.example.com`. Additionally, the *claimed attributes* can be found in the DNS at the name `_443._tcp.www.example.com.`. Given a "3 1 1" TLSA RRSset group, for example, the *claim* is quite simple: *one* of the presented values in the RRsset group contains the SHA2-256 digest of the target server's public key, *exactly*, and the public key is a value that can be received via a TLS handshake. The *verification* is also simple: first, connect to the hostname obtained from the base name of the RRsset's *name* (i.e. the DNS name without the `_443._tcp.`) and perform a TLS handshake with that server in order to receive its public key in DER format (or receive its end-entity TLS certificate and extract it); then, simply hash the public key using the SHA2-256 algorithm to obtain the relevant digest. If one of the TLSA RRssets in the "3 1 1" group is found, whose certificate data field is an exact match to this digest, then the server is successfully authenticated; otherwise, it is not. The verification process is, of course, a bit more involved with the other TLSA usages, as described in section 3.4.

The NIST standard about *Digital Identity Guidelines* (NIST SP 800-63-3) offers a definition for *digital authentication* as follows [104]:

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.

It further describes the authentication process, which includes the important first step of "*the claimant demonstrating to the verifier possession and control of an authenticator that is bound to the asserted identity*".

The NIST standard focuses more on authenticating and authorizing *users* rather than *servers*, but it offers the valuable insight that during authentication, the verifier establishes whether the entity, who claims to be a specific identity, is in *control* of an authenticator (i.e. a “token”, such as a digest). Control is demonstrated via DNSSEC: the DNS zone owner owns their public and private keys for their zone, and trust in these keys is established via the chain of trust described in section 3.3. Only a signed zone is trusted; only with those private keys, it is possible to sign the zone; only the DNS zone owner knows those private keys.

The ISO standard has more detailed guidelines attached to its definition of “**verification**” [101]:

Verification typically involves determining which attributes are needed to recognize an entity in a domain, checking that these required attributes are present, that they have the correct syntax, and exist within a defined validity period and pertain to the entity

RFC 6125 provides the following guideline about *authentication* [103]:

In general, a client needs to verify that the server’s presented identity matches its reference identity so it can authenticate the communication.

RFC 6125 also provides and aggregates other important concepts and general guidelines about authentication. For example, it references RFC 2818, which contains the following guideline on how name checks against a TLS certificate should be performed:

If a subjectAltName extension of type dNSName is present, that MUST be used as the identity. Otherwise, the (most specific) Common Name field in the Subject field of the certificate MUST be used. Although the use of the Common Name is existing practice, it is deprecated and Certification Authorities are encouraged to use the dNSName instead.

4.2 Identity life cycle and identity management

Even if the term “*identity*” has been clarified, there is still one *more specific* definition pertaining to the system that we desire to establish: “**identity management**” (or “**IdM**” for short).

ISO/IEC 24760-1 provides the definition [ISO24760-1]:

Processes and policies involved in managing the lifecycle and value, type and optional meta-data of attributes in identities known in a particular domain.

The *life cycle* defined in the ISO standard is shown in figure 4.2. It is the state diagram of an identity that starts in the *Unknown* state and becomes known by *enrollment*; it then goes through the process of *activation*, optionally *suspension* and *reactivation*. An identity manager can later *archive* or *delete* the identity. The states change back and forth when one wishes to change the identity. The rest of

this section will look at each transition in detail and apply it to the identity which emerges from a DNS name and the DANE TLSA association corresponding to a TLS certificate.

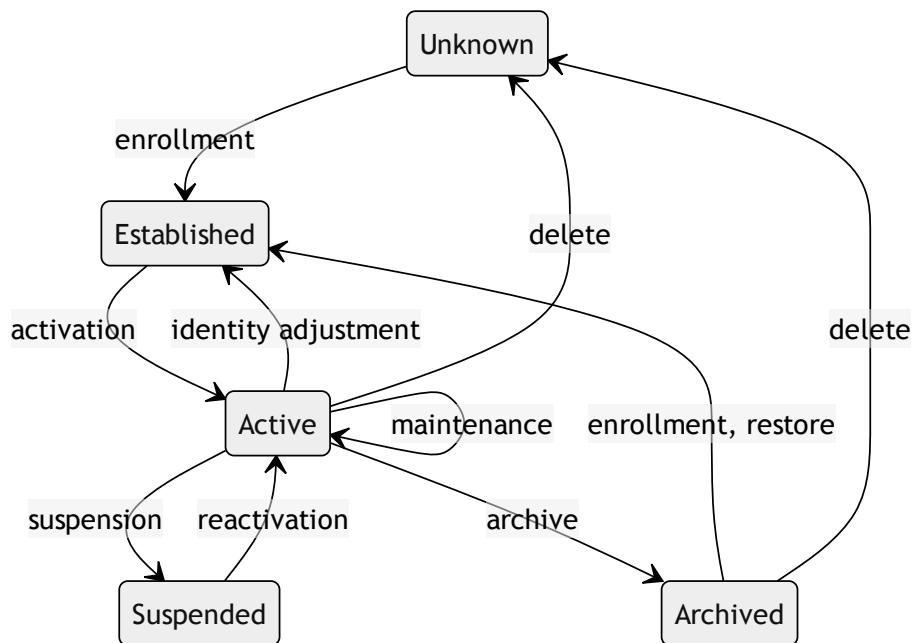


Figure 4.2: Identity life cycle

Life cycle event considerations for a DANE TLSA identity

All states and transitions are explained in detail in ISO/IEC 24760-1 [101]. We can now apply this standard to DANE identities.

When a DANE identity is being **enrolled**, all required attributes must be specified before the identity can be **established**. The entity attributes enumerated in section 4.1 must be provided. Some attributes can be derived, some attributes need to be provided by the user. The data provided by the user also needs to be verified in a process known as *identity proofing*. As soon as all information necessary for automatically managing the DANE records and TLS certificates has been provided and verified, the identity has been **established**.

Activation is the process of adding the established identity to the list of identities which are *actively used* for the core purpose. For DANE, the core purpose is server authentication, so an **active** identity would be one with a *currently used* TLS certificate and one or more *functioning, deployed* TLSA RR associated with it. A DANE client is expected to validate the server using an active identity.

Suspension requires the identity to be unusable, which can be reversed with **reactivation**. A **suspended** identity must be indicated as such. **Archival** is the reversible removal of the identity from the identity manager. An **archived** identity exists for statistical purposes, e.g. logging, but can also serve the identity proofing process during **restoration**. An archived identity is also useful to prevent conflicts with an identity about to be enrolled. For example, reusing a hostname is theoretically possible, but certain DNS providers or TLS providers might disallow it. The DANE identity manager must ensure that registering a hostname will succeed in both providers, even if the hostname has not been used before. Finally, **deletion** is the permanent removal of an identity from the identity manager.

Since generating TLSA records is trivial, a record can simply be removed from the DNS and, later, generated anew. TLS certificates can also easily be re-issued and renewed. An expired or revoked certificate can simply be replaced by a renewed certificate within the same identity, which does not require archival or suspension of the DANE identity, because this is a normal part of a *TLS certificate life cycle*. Deletion of DANE identities is useful if one wishes to discontinue a service. Removal of DANE *support* can be thought of as a suspension, especially regarding the requirement of a *functioning, deployed* TLSA RR in the *active* state.

However, the *TLS certificate* would still need to be in use, even if DANE support is switched off (for whatever reason). This calls for a separation between DANE identity management and TLS identity management. Similarly, since various DNS entries are part of the identity of a *server*, their changes must also be considered, which calls for a separation between DANE identity management and DNS identity management. Obviously, the management of DNS, of TLS, and of DANE are all intertwined, because TLS and DNS management needs to be synchronized in order for DANE to work properly. Fortunately, there are plenty of existing DNS managers and TLS certificate managers; the challenge relevant here is to create a system, which uses DANE identities as a basis, and communicates with two *subsystems*—one being a DNS manager, the other being a TLS certificate manager, both of which might have their own identity interpretation.

DANE identity adjustment and maintenance: the “state transitions” of a TLSA entry

In the life cycle shown in figure 4.2, there are two state transitions whose DANE-related interpretation is a bit more involved. **Identity adjustment** and **maintenance** both involve changes in the identity attribute. The difference between them is that the *identity adjustment* is a change that justifies a change in the activation state. In this subsection, we will look at possible changes to the identity attributes and determine what needs to be done in these cases.

If the *current* certificate changes, be it due to expiration or revocation, then a new TLSA RR could be generated, but does not always need to be. For example, if all affected TLSA RRs only have the selector 1 (SPKI) and the public key did not change across certificate replacement, then nothing needs to be done. If, however, any TLSA RR will change due to any kind of certificate change (or if a new record will be created), then updating the TLSA RR needs to be coordinated carefully. Similarly, if one decides to change the desired TLSA parameters for any certificate, invoking some change in existing TLSA RRs, updating them needs to be done carefully.

The following state transitions are covered by section 4.3 and section 8 in RFC 7671 and described there [80]. Note that these procedures only need to be performed if a change in the TLS certificate would cause a change in the TLSA RR. Also, note that publishing duplicate DNS records is implicitly ignored.

Switching between “DANE-” and “PKIX-” usages. Switching between the usages PKIX-TA (0) and DANE-TA (2), or between PKIX-EE (1) and DANE-EE (3) is motivated due to a possible future recommendation that one usage should be preferred over the other. During adoption of this recommendation, it is expected that some clients will still only support the less preferred set of usages. A TLSA publisher should therefore gradually switch from one usage to the other by first adding the newly preferred usage. Once enough clients have switched over to the preferred usage, the TLSA publisher can eventually remove the old usage.

Adding a usage *uNew* for a certificate *cert* means:

1. Let *tlsaRecordsOld* be the list of all TLSA RRs that correspond to certificate *cert*.
2. For each *tlsaRecordOld* of *tlsaRecordsOld*,
 1. let « *u*, *s*, *m* » be the parameter combination of *tlsaRecordOld*,
 2. let *tlsaRecordNew* be a new TLSA RR generated from *cert* with the desired parameter combination « *uNew*, *s*, *m* »,
 3. publish *tlsaRecordNew*.

The name and TTL of the new RR can be the same; the parameter combination will simply be different. In this case, the certificate data does not necessarily change. The algorithm for generating a new TLSA RR is described in section 3.4. A lot of time can pass between the addition of the new usage and removal of the old usage, the point of which is to not cause any disruption of clients being able to use the existing set of TLSA RRs for validation.

Server key rollover or certificate change with fixed TLSA parameters. This is where DNS caching becomes relevant: a new TLSA RR cannot simply *replace* the old one. Instead, some time needs to pass (at least twice the TTL) during a transitional state in which a TLSA for the new key and one for the old key needs to be published simultaneously. The RFC only specifies the key rollover together with the recommendation of publishing a “3 1 1” TLSA RR. If only the certificate got renewed, without changing the public key, this would not involve a change in any such TLSA RR. However, given the general publisher requirements from section 3.4, we can generalize the following process to a replacement of the certificate as well. For example, this should work for “3 0 1” TLSA RR, which contain a digest of the full certificate.

Switching from the existing certificate *certOld* to the new certificate *certNew* involves the following steps:

1. Let *tlsaRecordsOld* be the list of all TLSA RRs that correspond to *certOld*.
2. Let *tlsaRecordsNew* be an empty list.
3. For each *tlsaRecordOld* of *tlsaRecordsOld*,
 1. let « *u*, *s*, *m* » be the parameter combination of *tlsaRecordOld*,

2. let *tlsaRecordNew* be a new TLSA RR generated from *certNew* with the desired parameter combination « *u*, *s*, *m* »,
3. append *tlsaRecordNew* to *tlsaRecordsNew*,
4. publish *tlsaRecordNew*.
4. Wait at least 2× TTL.
5. For each *tlsaRecordNew* of *tlsaRecordsNew*,
 1. assert that *tlsaRecordNew* is now populated in DNS caches.
6. Publish *certNew*.
7. Assert that TLS server now serves *certNew*.
8. For each *tlsaRecordNew* of *tlsaRecordsNew*,
 1. delete *tlsaRecordNew*.

Switching TLSA parameters in general. This use case is only an addendum to the previous use case of a server key rollover. If one wishes to rotate the server key while switching to different TLSA parameters, the RFC explains this briefly:

In this case, publish the new parameter combinations for the server’s existing certificate chain first, and only then deploy new keys if desired.

Switching from “1 1 1” to “3 1 1” is the example provided by the RFC, however the RFC is not perfectly clear here. In the end, this works out fine, because publishing the TLSA RR with the new parameter combinations indicating the new usage will not “break” validation. The key rollover involves the *waiting* step which guarantees that caches contain the fresh data. This process also does not contradict the first state transition (*Switching between “DANE-” and “PKIX-” usages*), because the motivation is different: here, we simply want to replace the usage right away (but with the required minimum waiting time), with no regard to client support; in the other case, we wanted to gradually migrate to a new recommendation, while still supporting as many clients as possible.

Switching TLSA parameters from “3 1 *” to “2 0 *” while migrating to a new certificate chain. Specifically, the RFC explains a relatively complicated change with the following example. A given DANE identity *currently* uses

- a self-signed server certificate,
- a “3 1 1” TLSA record, and
- a specific public key

After the change, the DANE identity would use

- a new certificate chain,
- a “2 0 1” TLSA record, and
- a new public key

The RFC mandates that the change should involve a transitional “3 1 1” TLSA record, which needs to be populated into DNS caches, before deploying the new certificate chain and subsequently replacing the old TLSA RRs with the new, desired, “2 0 1” TLSA record. If a new “2 0 1” TLSA record is introduced right away, then the old “3 1 1” TLSA record might still persist in a DNS cache, which would cause the “2 0 1” TLSA record to correspond to the *new* certificate chain, whereas the “3 1 1” TLSA record would correspond to the *old* public key. This would violate the requirement of every TLSA parameter combination group to be sufficient to validate the TLS server, as discussed in section 3.4.

The reason why the transitional TLSA record must have the parameter combination “3 1 1” is that clients might want to negotiate the exchange of raw public keys, based solely on the *presence* of a “3 1 1” TLSA record. This is not possible with a certificate corresponding to a “2 0 1” TLSA record.

Therefore, these are the steps for switching from the existing certificate *certOld*, the usage 3, and the selector 1 to the new certificate *certNew*, the usage 2, and the selector 0:

1. Let *tlsaRecordsOld* be the list of all TLSA RRs that correspond to *certOld*, have usage 3, and selector 1.
2. Let *tlsaRecordsTransitional* be an empty list.
3. For each *tlsaRecordOld* of *tlsaRecordsOld*,
 1. let *m* be the matching type of *tlsaRecordOld*,
 2. let *tlsaRecordTransitional* be a new TLSA RR generated from *certNew* with the desired parameter combination « 3, 1, *m* »,
 3. append *tlsaRecordTransitional* to *tlsaRecordsTransitional*,
 4. publish *tlsaRecordTransitional*.
4. Wait at least 2× TTL.
5. For each *tlsaRecordTransitional* of *tlsaRecordsTransitional*,
 1. assert that *tlsaRecordTransitional* is now populated in DNS caches.
6. Publish *certNew*.
7. Assert that TLS server now serves *certNew*.
8. For each *tlsaRecordOld* of *tlsaRecordsOld*,
 1. delete *tlsaRecordOld*.
9. For each *tlsaRecordTransitional* of *tlsaRecordsTransitional*,
 1. let *m* be the matching type of *tlsaRecordTransitional*,
 2. delete *tlsaRecordTransitional*,
 3. let *tlsaRecordNew* be a new TLSA RR generated from *certNew* with the desired parameter combination « 2, 0, *m* »,
 4. publish *tlsaRecordNew*.

Introducing a new matching type. Section 3.4 mentions the requirement to only publish TLSA records with matching type 2 if a TLSA record with matching type 1 and the same usage and selector exists as well. There is also a requirement about *algorithm agility* which states that if a TLSA record with matching type 2 and a specific usage and selector exists, then TLSA records must also exist *for all other* usages and selectors, with matching type 2. A matching type must either entirely exist across all TLSA records or entirely *not* exist across all TLSA records. It cannot be used for *some but not all* TLSA records.

Therefore, enabling the new matching type m , these steps need to be performed:

1. Let *certs* be the list of all currently used certificates.
2. For each *cert* of *certs*,
 1. Let *tlsaRecords* be the list of all TLSA RRs that correspond to *cert* and do not have matching type m .
 2. For each *tlsaRecord* of *tlsaRecords*,
 1. let u be the usage of *tlsaRecord*,
 2. let s be the selector of *tlsaRecord*,
 3. let *tlsaRecordNew* be a new TLSA RR generated from *cert* with the desired parameter combination « u, s, m »,
 4. publish *tlsaRecordNew*.

Removing a specific matching type. This applies the inverse of the same rule as stated above.

To disable the matching type m , these steps need to be performed:

1. Let *tlsaRecords* be the list of all TLSA RRs that have matching type m .
2. For each *tlsaRecord* of *tlsaRecords*,
 1. delete *tlsaRecord*.

Note that, in general, the step “Wait at least $2 \times TTL$ ” is not needed during *initial* activation.

Now, the question is: are these *identity adjustments* or *maintenance*? First, the step “Wait at least $2 \times TTL$ ” might suggest that something is happening which makes the current certificate association temporarily unusable. But every process is carefully crafted to *avoid* this; otherwise, it might cause a temporary outage of the Web service. The waiting does not actually impact the identity. Certificate expiration might cause an outage, though not with usage 3 (DANE-EE), and certificate revocation might play a role. However, revocation does not imply immediate invalidity of the certificate. Server administrators still have some time to replace a revoked certificate, and they have the option to supply a replacement in advance—all this can also be automated. And expiration of the current certificate should never realistically happen if renewal is automated, because it is good practice to renew the certificate several weeks before its expiration [105]. The only scenario in which a certificate becomes unusable, with no replacement issued, is if the identity management application is unavailable for several weeks. Such a disastrously exceptional scenario is where the life cycle and identity management model breaks down, regardless.

None of the transitions described above justify temporarily deactivating the DANE identity, even if all of them require changes in the identity attributes. One change that might be considered identity adjustment, and therefore justify deactivation, would be the manual full replacement of a certificate, e.g. due to a change in the identifying information embedded in the certificate. Since removing obsolete TLSA records is recommended by RFC 7671, any identity adjustment would start with this. But since TLSA records are kept in DNS caches for a while, they will correspond to the certificate that is about to be removed. Since this would cause DANE authentication to fail, this case is not allowed to happen. Instead, the manually replaced certificate should only be *scheduled* to be deployed; then the identity manager will simply follow the same steps as described in *Server key rollover or certificate change with fixed TLSA parameters*. It is possible that no *identity adjustment* ever needs to occur.

Other DANE bindings—IPSECKEY, SSHFP, OPENPGPKEY, and SMIMEA—would have their own identities, i.e. “tracks” that keep track of the *currently used* DNS records and certificate or public key, but this thesis only focuses on the TLSA binding for now.

5 User-centered design implementation using web technologies

This section includes a few concrete ideas for an implementation of the identity manager to be established. The purpose of the identity manager is to automate the TLSA publishing process and help avoid user errors. A user-centered design approach shall provide an intuitive workflow in which the user is guided through the process. We will look at a few helpful tools and briefly discuss the back end and the front end design.

5.1 Tools

There are a few helpful cryptography- and network-related tools that can be used in an identity manager.

DNS look-up utilities

`host` [106] and `dig` [107] are two DNS look-up utilities. `host` is a simple utility for quick look-ups whereas `dig` is a more flexible tool useful in debugging.

A simple command like this reveals the IP addresses associated with a domain name:

```
host 'www.example.com.'
```

```
www.example.com has address 93.184.216.34
```

```
www.example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
```

With one of the following commands, it is possible to look up all RRs of the `example.com` DNS zone from the name server configured on the system:

```
host -a 'example.com.'
```

```
dig 'example.com.' 'ANY'
```

A different name server can be specified; the one in the NS RR of `example.com` (`a.iana-servers.net`) is the authority for that domain name and should provide an *authoritative answer* (with the *aa* bit set, as specified in RFC 1035 [69]).

```
host 'example.com.' 'a.iana-servers.net.'
dig '@a.iana-servers.net.' 'example.com.'
```

The parent zone is simply `com.`, the root zone is `.` (the dot). Both of these can easily be queried; the `-t` flag can be used to query RRs of a specific type:

```
host -a 'com.'          # Welcome to the `com.` zone!
host -t 'DNSKEY' '.'    # Public keys of the root zone;
                        #   the one with flag 256 is the ZSK
                        #   the one with flag 257 is the KSK.
```

The `www` name is commonly used for the web service of a domain. In order to reach the TLSA RR of the web service, the protocol and port need to be specified as well, forming the complete DNS name of `_443._tcp.www.example.com`. A lookup of these RRs would be performed with one of these commands:

```
host -t 'TLSA' '_443._tcp.www.example.com'
dig '_443._tcp.www.example.com' 'TLSA'
```

Alternatively, the domain name can be dynamically constructed:

```
port='443'
protocol='tcp'
domain='www.example.com.'

host -t 'TLSA' "_${port}.${protocol}.${domain}"
```

The real `example.com` does not currently support DANE; in this case we get a response involving status: `NXDOMAIN` and a single SOA RR, indicating that the name `_443._tcp.www.example.com` does not exist. This is the example output for `www.huque.com`¹:

```
dig '_443._tcp.www.huque.com' 'TLSA'

; <<>> DiG 9.18.13 <<>> _443._tcp.www.huque.com TLSA
;; global options: +cmd
;; Got answer:
;; -->HEADER<<- opcode: QUERY, status: NOERROR, id: 59501
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 8, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:;; udp: 512
;; QUESTION SECTION:
;_443._tcp.www.huque.com.    IN    TLSA
```

¹The “ANSWER SECTION” contains eight long lines which have been wrapped to fit into the page.

```
;; ANSWER SECTION:
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
7634E0EED7DFCA738733ED09C8BED00D4C5F4A3E15F76BC9E0B6E57F DBB9163D
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
6C85CC093C31221CBFF9E61CFF5E9CA14BFEB0F9BBC341A769529027 5D813CF4
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
CD98C1661E982DA43E414E9EEB7A3A545DEEC8859F5282E28CA3CBB3 82DDBE3E
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
33F5B0C36FBDE25D8B148623DEB22A4CD5D1A806EE1259F48DD9ABDF 36EE8A63
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
DE4369CF0866A1E7626D73DB36DBFC4B74097C3C70489A2D3351B6E7 5E99583A
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
A67924AFCD895B9661C4C5D67A83215F60B7D0E1DA8A30B67EEC6BD6 23A5B57C
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
244B484761B1CC2A390DB3F1031C1D512AF3C9C14DF47854E43E3C29 17EF671C
_443._tcp.www.huque.com. 7169 IN TLSA 3 1 1
EE7041856CC8AD23789D39462A7D11C0B7633C6142ADE119BC9B1BEF 708F7D0A

;; Query time: 16 msec
;; SERVER: 192.168.178.1#53(192.168.178.1) (UDP)
;; WHEN: Mon Apr 17 16:08:24 CEST 2023
;; MSG SIZE rcvd: 428
```

TLS look-up utility

nmap [108] is a network and security utility. The following command can extract the TLS certificate currently used by a host [109]:

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

Cryptography, certificate management

OpenSSL [110] and Certbot [111] are two very versatile tools.

The appendix shows how one can use the subcommands `openssl genpkey`, `openssl rsa`, `openssl pkeyutl`, `openssl req`, and `openssl verify` in order to create and verify certificates.

`openssl s_client` provides an SSL/TLS client, which even provides DANE authentication using the `-dane_tlsa_domain` and `-dane_tlsa_rrdata` options. This can be used to verify correct deployment of TLS certificates on the server and correct matching of TLSA records.

Finally, `openssl x509` and `openssl enc` can be used to extract public keys and transcode PEM into DER format, respectively. Paired with basic tools such as `sha256sum` or `sha512sum`, `cut`, and `tr`, one can generate a TLSA RR as described in section 3.4.

```
openssl 'x509' -in 'huque.pem' -pubkey -nocert -noout |
  openssl 'enc' -d -base64 -out 'huque-public.der' &&
  sha256sum 'huque-public.der' |
  cut --delimiter=' ' --fields='1' |
  tr '[:lower:]' '[:upper:]'
```

5.2 Back end

Conceptionally, a single system is required that has access to both DNS zone management as well as TLS certificate management for the same domain. These two management “subsystems” might either be integrated into the same code base or communicate via an API. The DNS management subsystem would have all the normal functionality of existing DNS managers, i.e. adding, removing, editing records, etc. The TLS management subsystem would facilitate the automatic issuing and renewal of TLS certificates, as well as handle revocations—this also exists. The crucial exception for the DNS manager is to *not allow* direct manipulation of TLSA, IPSECKEY, SSHFP, OPENPGPKEY, and SMIMEA records; this is the job of the DANE identity manager.

For example, if the TLS manager wants to renew a certificate, it would first notify the identity manager, which would in turn notify the DNS management subsystem. A new TLSA RR will be created; the DNS manager needs to be notified about the request to publish it. The TLS manager needs to wait for its signal to publish the certificate. Figure 5.1 presents this rough idea visually.

Since a certificate can be issued for a wildcard domain (e.g. `*.example.com`), it is useful to explicitly enumerate the DNS names the certificate applies to, so that TLSA RR names can be more easily identified.

A DANE identity manager should log every step of any DNS or TLS management operation, which is an optional requirement from the ISO/IEC 24760-1 standard [101]. This might aid in auditing and fault recovery. In case the identity manager itself suffers an outage, as described in section 4.2, a few steps could be taken to mitigate the lack of *management* after it becomes available again. For example, the identity manager could audit all DANE identities and their current states, and

- validate their integrity,
- renew certificates which are revoked, expired, or about to expire,
- test if they are usable, i.e.
 - check if the certificate is being correctly served from the server and
 - check if the DANE records are deployed in the DNS,
- act as a DANE client and perform an authentication.

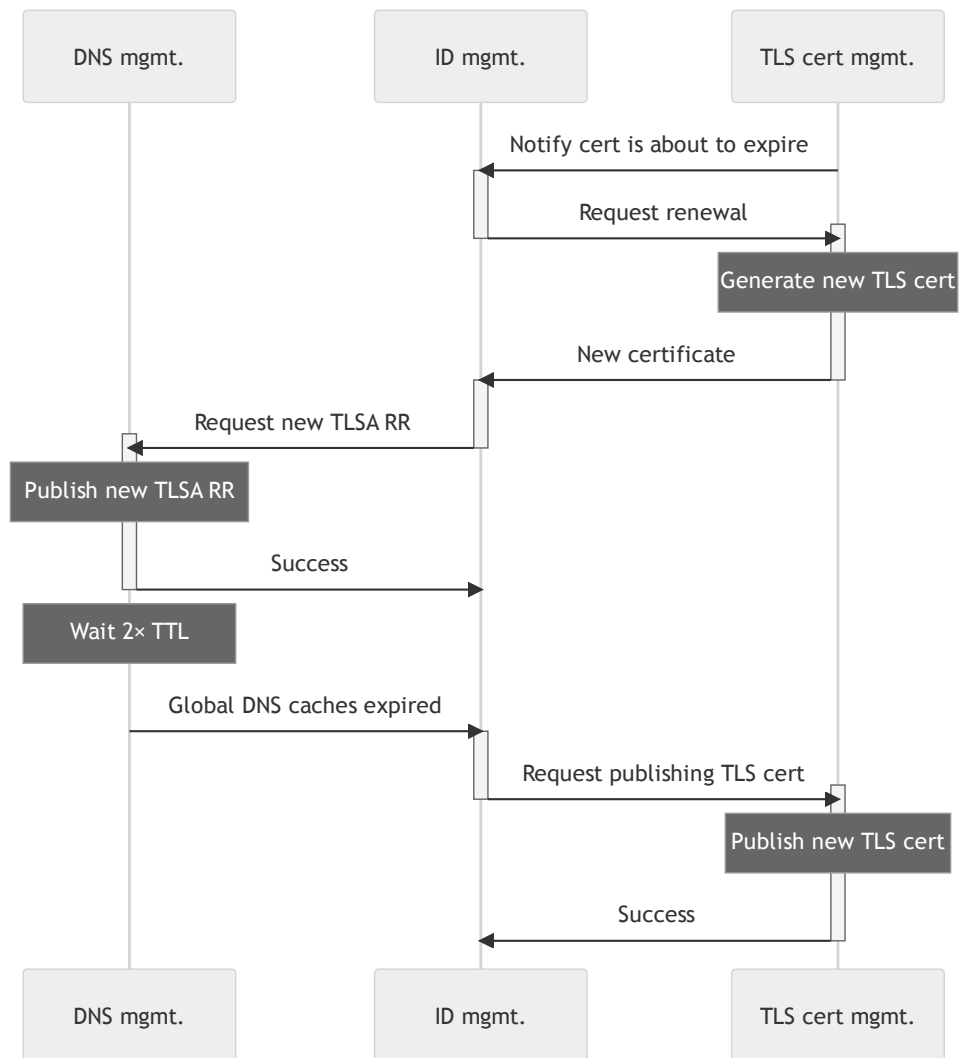


Figure 5.1: Example communication sequence of publishing TLSA RRs while renewing TLS certificate

These checks could also occur periodically or before and after critical operations.

There should also be a way for the identity manager to accept an *existing* identity state; for example, if TLSA records already exist in a zone, they could be imported to the new identity manager, or they could be exported to some other identity manager. Ideally, this should also be possible during an outage.

Some steps can be taken to *future-proof* TLSA identity management, so that changes or extensions in the DANE TLSA standards can be easily adapted to in the future:

- Matching type: Algorithm agility (also, crypto agility) is recommended [80]. Currently, only SHA2-256 and SHA2-512 are supported, but common digest algorithms used in related cryptography protocols can be prepared, even if they're unlikely to ever be included in the TLSA standard. After all, digest algorithms being discovered to be “broken” is not unprecedented.
- Selector: Unlikely to ever change, but an X.509 parser and DER encoder (i.e. OpenSSL) should be available to a TLSA publisher to extract various fields from a certificate, just in case.
- Usage: Seems to be the most versatile since it already encodes *two* pieces of information in a single field (2 bits). It's plausible that extensions to PKIX or entirely new PKI approaches force a rethinking of how authentication needs to be performed. The third bit, fourth bit, and so on, could be used for other decisions—perhaps unforeseeable today. RFC 6698 states [4]:

The certificate usages defined in this document explicitly only apply to PKIX-formatted certificates in DER encoding (X.690). If TLS allows other formats later, or if extensions to this RRtype are made that accept other formats for certificates, those certificates will need their own certificate usage values.

- A new fourth field: unlikely to be backward-compatible. It's more likely that anything *entirely new* will be “crammed” into the Usage field, i.e. the Usage field being repurposed for new modes of TLSA authentication. Alternatively, a new record type will be established and TLSA deprecated.

5.3 Web application and user interface

Since the term “identity” is rather complex, it should be a concept that is only handled by the back end; the user should not be confronted with this concept. DNS zone management and certificate management can be regarded as two “subsystems” of DANE identity management; these are more likely to be familiar to DNS administrators and server administrators. Therefore, the user interface would involve these two subsystems.

Here, the layout used by the deSEC prototype might be useful: one tab for DNS management, another for Identity management [34]. Since the other four DANE bindings should be considered as

well, the second tab should not just be labeled “TLS management”—instead, “Identity management” is a suitable alternative. This tab would then have the five subtabs corresponding to each DANE binding. This reflects the two components that constitute the system: *DNS management* and *key or certificate management*.

The DNS manager would be a simple editable table of Resource Records. DANE-specific resource records would be marked specially, and would only be able to be edited *indirectly* via the identity manager.

The key or certificate manager is the more interesting component. The “TLS” subtab would show a table, each row corresponding to an identity, i.e. a “track” of certificates. The columns would correspond to the identity attributes from section 4.1:

- **Label:** human-readable identifier
- **Hostnames:** list of hostnames that the current certificate applies to
- **Valid from**
- **Valid until**
- **Type:** List of usage-selector combinations for generating the TLSA RRs
- **Status:** deployment state of the current certificate and the TLSA RRs

The *list* referred to in the “Type” column could also be empty. This would put the identity in the “No DANE” state.

The “Status” could be one of the following:

- **Inactive:** user will be prompted to configure and enable once; guide them (corresponds to “Established”)
- **No DANE:** if no DANE records will be deployed for the time being
- **Scheduled:** if new certificate is ready to be deployed, but the TLSA requires the necessary waiting period (corresponds to “Established” or “Active”, depending on whether TLSA RRs already have deployed)
- **Deployed:** happy state (corresponds to “Active”)
- **Revoked:** refers to the *identity* being revoked (corresponds to “Archived”)
- **Error:** user will be prompted for action and informed via e-mail

Clicking on one of the identity rows displays all the above pieces of data and additionally:

- A stored certificate signing request (CSR)
- Authentication token for a CA
- The *current* TLSA RRs that correspond to the certificate
- References to future or past certificates during a rollover

The matching type will not be a per-identity option, but a globally configurable list of enabled hashing algorithms; of course, SHA2-256 can never be disabled.

The key or certificate manager would allow adding new identities of the desired DANE binding in the corresponding subtab. For example, the user clicks on the “Identity manager” tab, then

navigates to the “TLS / TLSA” subtab. To create a new identity, the user would click on an “Add” button, upload a certificate via another button, or via drag & drop, or via some other process to generate certificates on the fly. A dialog will prompt the user for the required identity attributes. The identity manager will then suggest the TLSA record policy based on the uploaded certificate; the result is a *list* of suggestions

- If the certificate is an end-entity certificate, suggest only usage DANE-EE, selector SPKI, and matching type SHA2-256
- Else, if the certificate is an end-entity certificate, suggest only usage DANE-TA, selector Cert, and matching type SHA2-256
- If SHA2-512 is enabled globally, add these suggestions to the list.

The user should still be able to add and remove other options. Later, when editing the identity, this list can always be adjusted, including removing all items. The identity manager should always warn the user in case they select an option that is not recommended, as per section 3.4. Editing the list of usages and selectors, as well as the global list of matching types, invokes the algorithms for state transitions of TLSA entries, described in section 4.2.

The hostnames can be automatically extracted from the certificate.

- If the certificate is issued for `a.example.com` and `b.example.com`, create TLSA entries at `_443._tcp.a.example.com` and `_443._tcp.b.example.com`. Alternatively, coalesce them into a single TLSA record with a name including `_dane`, using indirection via CNAME, as per section 3.4—perhaps using the human-readable label as well.
- If the certificate is issued for `*.example.com`, create corresponding TLSA entries for every DNS name that is *currently in use*, keeping track of future updates.

When viewing one specific identity, the user will have options for different actions:

- **Remove:** terminates the identity, archives it, deletes all corresponding TLSA RRs.
- **Replace:** schedules new certificate to replace the current one with, as soon as the TLSA RR update allows.

In the appendix, some of these ideas are visualized.

6 Conclusion and future work

DANE provides an opportunity for a better trust model—if implemented correctly. Time will tell if it really turns out to be a better option than the current public-key infrastructure model for TLS.

We show that management can be simplified for DNS zone administrators. The prototyped application follows the protocol specification as well as identity management standards. The proposed ideas for an intuitive user interface and automation in the back end should contribute greatly to decreasing human error and thus be beneficial for security.

In section 2.1 and section 2.3, a lot of doubt about DNSSEC and DANE can be seen by protocol implementers and client users. Since a significant portion of this doubt stems from the high complexity involved in managing DNSSEC and DANE, hopefully, this thesis can serve as a useful starting point for future implementation efforts. Section 5, especially, proposes how a possible integration into a real DNS management system might look like, simplifying implementation efforts for domain registrars, and, in turn, streamlining the process for DNS administrators, possibly leading to higher adoption rates in the near future. The chicken-and-egg problem might start getting solved using this research.

However, implementers will still need to review the referenced standards and protocol specifications carefully. Certain considerations, unfortunately, did not fit into the scope of this thesis; it is recommended that future research keeps these in mind. For example:

- More complex hosting setups, e.g. DANE used with delegated subzones (e.g. RFC 7671)
- *A framework for identity management—Part 2: Reference architecture and requirements* (ISO/IEC 24760-2)
- General information security guidelines (e.g. ISO 27001)
- DANE integration with SRV RRs (RFC 7673)
- Usability guidelines (e.g. ISO 9241, ISO/IEC 25066)
- Accessibility guidelines (e.g. WAI-ARIA)
- User testing
- User documentation
- Developer documentation
- Atomicity
- Interoperability
- Internationalization

Further research into possible roadblocks during implementation is desired, as well as an evaluation in reference to this thesis. In tandem, registrar support for DANE and deployment of DANE in real domains need to be continually measured—both in quantity and quality.

References

- [1] P. Mockapetris, “Domain names: Concepts and facilities.” RFC 882, Nov. 1983 [Online]. Available: <https://www.rfc-editor.org/info/rfc882>
- [2] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, “DNS Security Introduction and Requirements.” RFC 4033, Mar. 2005 [Online]. Available: <https://www.rfc-editor.org/info/rfc4033>
- [3] ITU, “Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks.” X.509; International Telecommunication Union Telecommunication Standardization Sector (ITU-T), Oct. 2019 [Online]. Available: <https://www.itu.int/rec/T-REC-X.509-201910-I/en>
- [4] P. E. Hoffman and J. Schlyter, “The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA.” RFC 6698, Aug. 2012 [Online]. Available: <https://www.rfc-editor.org/info/rfc6698>
- [5] D. Atkins and R. Austein, “Threat Analysis of the Domain Name System (DNS).” RFC 3833, Aug. 2004 [Online]. Available: <https://www.rfc-editor.org/info/rfc3833>
- [6] J. Stewart, “DNS cache poisoning—the next generation.” 2003 [Online]. Available: https://www.thefengs.com/wuchang/courses/old_courses/cs510netsec_fall2005/summaries/10.pdf
- [7] M. Müller, J. de Jong, M. van Heesch, B. Overeinder, and R. van Rijswijk-Deij, “Retrofitting post-quantum cryptography in internet protocols: A case study of DNSSEC,” *SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 4, pp. 49–57, Oct. 2020, doi: 10.1145/3431832.3431838. [Online]. Available: <https://doi.org/10.1145/3431832.3431838>
- [8] S. Kim, S. Lee, G. Cho, M. E. Ahmed, J. (Paul). Jeong, and H. Kim, “Preventing DNS amplification attacks using the history of DNS queries with SDN,” in *Computer security – ESORICS 2017*, 2017, pp. 135–152 [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-66399-9_8
- [9] S. Saharan and V. Gupta, “Prevention and mitigation of DNS based DDoS attacks in SDN environment,” in *2019 11th international conference on communication systems & networks (COMSNETS)*, Jan. 2019, pp. 571–573, doi: 10.1109/COMSNETS.2019.8711258 [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8711258>
- [10] IANA, “DNSSEC project archive – launch status updates.” Jul. 2020 [Online]. Available: <https://www.iana.org/dnssec/archive/launch-status-updates>. [Accessed: Apr. 17, 2023]

- [11] T. Chung *et al.*, “Understanding the Role of Registrars in DNSSEC Deployment,” in *Proceedings of the 2017 internet measurement conference*, 2017, pp. 369–383, doi: 10.1145/3131365.3131373.
- [12] T. Chung, “Why DNSSEC deployment remains so low.” Dec. 06, 2017 [Online]. Available: <https://blog.apnic.net/2017/12/06/dnssec-deployment-remains-low/>. [Accessed: Apr. 17, 2023]
- [13] N. L. M. van Adrichem *et al.*, “A measurement study of DNSSEC misconfigurations,” *Security Informatics*, vol. 4, no. 1, p. 8, Oct. 2015, doi: 10.1186/s13388-015-0023-y.
- [14] L. Constantin, “DNSSEC explained: Why you might want to implement it on your domain.” Jul. 30, 2020 [Online]. Available: <https://www.csoonline.com/article/3569277/dnssec-explained-why-you-might-want-to-implement-it-on-your-domain.html>. [Accessed: Apr. 17, 2023]
- [15] IANA, “Key signing ceremonies.” 2023 [Online]. Available: <https://www.iana.org/dnssec/ceremonies>. [Accessed: Apr. 17, 2023]
- [16] S. Roth, R. Rijswijk-Deij, and T. Chung, “Tracking registrar support for DNSSEC: It’s slowly getting better,” in *Proceedings of the internet measurement conference*, Oct. 2019, doi: 10.1145/3355369.3355596 [Online]. Available: <https://taejoong.github.io/pubs/publications/spencer-2019-dnssec.pdf>. [Accessed: Apr. 17, 2023]
- [17] Internet Systems Consortium, “Domain name system security extensions now deployed in all generic top-level domains.” Dec. 23, 2020 [Online]. Available: <https://www.icann.org/en/announcements/details/domain-name-system-security-extensions-now-deployed-in-all-generic-top-level-domains-23-12-2020-en>. [Accessed: Apr. 17, 2023]
- [18] F. Cambus, “List of ccTLDs.” 2023 [Online]. Available: <https://www.statdns.com/cclds/>. [Accessed: Apr. 17, 2023]
- [19] “Estimating IPv6 & DNSSEC Deployment Status.” NIST, 2023 [Online]. Available: <https://fedv6-deployment.antd.nist.gov/snap-all.html>. [Accessed: Apr. 17, 2023]
- [20] “DNSSEC world map.” APNIC Labs, 2023 [Online]. Available: <https://stats.labs.apnic.net/dnssec>. [Accessed: Apr. 17, 2023]
- [21] Wikipedia contributors, “Certificate authority — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Certificate_authority&oldid=1150391606, 2023 [Online]. Available: https://en.wikipedia.org/wiki/Certificate_authority. [Accessed: Apr. 19, 2023]
- [22] Infoblox, “What is DANE?” Infoblox company blog, 2020 [Online]. Available: <https://www.infoblox.com/dns-security-resource-center/dns-security-faq/what-is-dane/>. [Accessed: Apr. 19, 2023]
- [23] Wikipedia contributors, “DigiNotar — Wikipedia, the free encyclopedia.” <https://en.wikipedia.org/w/index.php?title=DigiNotar&oldid=1121079203>, 2022 [Online]. Available: <https://en.wikipedia.org/wiki/DigiNotar>. [Accessed: Apr. 19, 2023]

- [24] H. Adkins, “An update on attempted man-in-the-middle attacks.” Aug. 29, 2011 [Online]. Available: <https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html>. [Accessed: Apr. 17, 2023]
- [25] D. Huang and B. Hill, “Early impacts of certificate transparency.” Apr. 11, 2016 [Online]. Available: <https://www.facebook.com/notes/protect-the-graph/early-impacts-of-certificate-transparency/1709731569266987/>. [Accessed: Apr. 17, 2023]
- [26] A. Langley, “Maintaining digital certificate security.” Mar. 23, 2015 [Online]. Available: <https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html>. [Accessed: Apr. 19, 2023]
- [27] J. Leyden, “RIP HPKP: Google abandons public key pinning.” The Register, Oct. 30, 2017 [Online]. Available: https://www.theregister.com/2017/10/30/google_hpkp/. [Accessed: Apr. 19, 2023]
- [28] L. Tung, “Google: Chrome is backing away from public key pinning, and here’s why.” ZDNET, Oct. 30, 2017 [Online]. Available: <https://www.zdnet.com/article/google-chrome-is-backing-away-from-public-key-pinning-and-heres-why/>. [Accessed: Apr. 19, 2023]
- [29] S. Bortzmeyer, “DANE test sites.” Internet Society, 11710 Plaza America Drive, Suite 400, Reston, VA 20190, USA, Oct. 04, 2012 [Online]. Available: <https://www.internetsociety.org/resources/deploy360/dane-test-sites/>. [Accessed: Apr. 18, 2023]
- [30] H. Lee, A. Gireesh, R. van Rijswijk-Deij, T. ”Ted” Kwon, and T. Chung, “A longitudinal and comprehensive study of the DANE ecosystem in email,” in *29th USENIX security symposium (USENIX security 20)*, Aug. 2020, pp. 613–630 [Online]. Available: <https://www.usenix.org/conference/usenixsecurity20/presentation/lee-hyeonmin>
- [31] V. Dukhovni and W. Hardaker, “DNSSEC and DANE deployment statistics.” DNSSEC-Tools, May 09, 2016 [Online]. Available: <https://stats.dnssec-tools.org/>. [Accessed: Apr. 21, 2023]
- [32] V. Dukhovni, “Common mistakes.” sys4; dotplex, 2013 [Online]. Available: https://dane.sys4.de/common_mistakes. [Accessed: Apr. 18, 2023]
- [33] Xservus Limited, “DNSSEC – why not having a signed zone is almost never going to lead to you getting pwn3d.” PwnDefend, Feb. 15, 2023 [Online]. Available: <https://www.pwndefend.com/2023/02/15/dnssec-why-not-having-a-signed-zone-is-almost-never-going-to-lead-to-you-getting-pwn3d/>. [Accessed: Apr. 15, 2023]

- [34] P. Thomassen, “Retrieving and Creating DNS Records – Creating a TLSA RRset – deSEC DNS API documentation.” Via GitHub (<https://github.com/desec-io/desec-stack/blob/main/docs/dns/rrsets.rst>). Read The Docs, Jul. 28, 2017 [Online]. Available: <https://desec.readthedocs.io/en/latest/dns/rrsets.html#creating-a-tlsa-rrset>. [Accessed: Apr. 18, 2023]
- [35] N. Wisiol, “Identity manager for DNSSEC awesomeness.” GitHub, Feb. 01, 2021 [Online]. Available: <https://github.com/desec-io/desec-stack/pull/513>. [Accessed: Apr. 18, 2023]
- [36] GoDaddy Help, “Tweet by GoDaddy Help.” Twitter Thread; Twitter, Feb. 07, 2020 [Online]. Available: <https://twitter.com/GoDaddyHelp/status/1225875239122411520>. [Accessed: Apr. 18, 2023]
- [37] F. P. and Riko, “DNSSEC und DANE per default!” Support forum thread, Sep. 03, 2019 [Online]. Available: <https://forum.netcup.de/netcup-intern/operations/11803-dnssec-und-dane-per-default/>. [Accessed: Apr. 18, 2023]
- [38] O. de Sousa *et al.*, “DANE and TLSA.” Support forum thread, Dec. 21, 2013 [Online]. Available: <https://features.cpanel.net/topic/dane-and-tlsa>. [Accessed: Apr. 19, 2023]
- [39] sven2 and user19291, “Support DANE.” Support forum thread, Jan. 17, 2022 [Online]. Available: <https://community.cloudflare.com/t/support-dane/346464>. [Accessed: Apr. 18, 2023]
- [40] S. Huque, “Generate TLSA record.” [Online]. Available: https://www.huque.com/bin/gen_tlsa
- [41] V. Dukhovni, “DANE SMTP validator.” sys4; dotplex, Franziskanerstraße 15, 81669 Munich, Germany, 2013 [Online]. Available: <https://dane.sys4.de/>
- [42] “DANE validator.” SIDN Labs, 2013 [Online]. Available: <https://check.sidnlabs.nl/dane/>
- [43] S. Huque, “dane – Go library for DANE TLSA authentication.” GitHub repository; GitHub, May 26, 2020 [Online]. Available: <https://github.com/shuque/dane>. [Accessed: Apr. 18, 2023]
- [44] P. Wouters and D. Stöcker, “hash-slinger.” GitHub repository; GitHub, Sep. 15, 2012 [Online]. Available: <https://github.com/letoams/hash-slinger>. [Accessed: Apr. 18, 2023]
- [45] A. Langley, “Why not DANE in browsers.” Jan. 17, 2015 [Online]. Available: <https://www.imperialviolet.org/2015/01/17/notdane.html>. [Accessed: Apr. 19, 2023]
- [46] “Use DNSSEC/DANE chain stapled into TLS handshake in certificate chain validation.” Bugzilla Bug 672600; Bugzilla, Jul. 19, 2011 [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=672600. [Accessed: Apr. 18, 2023]
- [47] “Domain authenticated named entities against MITM-attacks.” Bugzilla Bug 1077323; Bugzilla, Oct. 03, 2014 [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=1077323. [Accessed: Apr. 18, 2023]

- [48] J. Aas, “Let’s encrypt: Delivering SSL/TLS everywhere.” Company blog, Nov. 18, 2014 [Online]. Available: <https://letsencrypt.org/2014/11/18/announcing-lets-encrypt.html>. [Accessed: Apr. 18, 2023]
- [49] defkev, “DNSSEC/DANE validator.” Firefox Browser Add-on; Mozilla, Apr. 08, 2021 [Online]. Available: <https://addons.mozilla.org/en-US/firefox/addon/dnssec-dane-validator/>. [Accessed: Apr. 18, 2023]
- [50] “Chromium does not support DANE validation on domains with DNSSEC enabled.” Chromium Issue 672600; Chromium Issue Tracker, Jun. 27, 2022 [Online]. Available: <https://bugs.chromium.org/p/chromium/issues/detail?id=1339991>. [Accessed: Apr. 19, 2023]
- [51] “Implement DNSSEC verification.” Chromium Issue 50874; Chromium Issue Tracker, Aug. 01, 2010 [Online]. Available: <https://bugs.chromium.org/p/chromium/issues/detail?id=50874>. [Accessed: Apr. 19, 2023]
- [52] S. Grüner, “DANE für browser ist praktisch tot.” May 28, 2019 [Online]. Available: <https://www.golem.de/news/dnssec-chain-dane-fuer-browser-ist-praktisch-tot-1905-141559.html>. [Accessed: Apr. 19, 2023]
- [53] T. Ptacek, “Against DNSSEC.” Jan. 15, 2015 [Online]. Available: <https://sockpuppet.org/blog/2015/01/15/against-dnssec/>. [Accessed: Apr. 19, 2023]
- [54] T. Ptacek, “Questions and answers from ‘Against DNSSEC’.” 2015 [Online]. Available: <https://sockpuppet.org/stuff/dnssec-qa.html>. [Accessed: Apr. 19, 2023]
- [55] Wikipedia contributors, “X.509 — Wikipedia, the free encyclopedia.” <https://en.wikipedia.org/w/index.php?title=X.509&oldid=1149564995>, 2023 [Online]. Available: <https://en.wikipedia.org/wiki/X.509>. [Accessed: Apr. 20, 2023]
- [56] R. W. Shirey, “Internet Security Glossary, Version 2.” RFC 4949, Aug. 2007 [Online]. Available: <https://www.rfc-editor.org/info/rfc4949>
- [57] Wikipedia contributors, “RSA (cryptosystem) — Wikipedia, the free encyclopedia.” [https://en.wikipedia.org/w/index.php?title=RSA_\(cryptosystem\)&oldid=1148626325](https://en.wikipedia.org/w/index.php?title=RSA_(cryptosystem)&oldid=1148626325), 2023 [Online]. Available: [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem)). [Accessed: Apr. 20, 2023]
- [58] ITU, “Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).” X.690; International Telecommunication Union Telecommunication Standardization Sector (ITU-T), Feb. 2021 [Online]. Available: <https://www.itu.int/rec/T-REC-X.690-202102-I/en>
- [59] S. Josefsson and S. Leonard, “Textual Encodings of PKIX, PKCS, and CMS Structures.” RFC 7468, Apr. 2015 [Online]. Available: <https://www.rfc-editor.org/info/rfc7468>
- [60] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.” RFC 5280, May 2008 [Online]. Available: <https://www.rfc-editor.org/info/rfc5280>

- [61] J. Sequeira and S. Ghazi, “OpenSSL tutorial.” University of Toronto, 2019 [Online]. Available: https://www.cs.toronto.edu/~arnold/427/19s/427_19S/tool/ssl/notes.pdf. [Accessed: Apr. 20, 2023]
- [62] Wikipedia contributors, “Public-key cryptography — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Public-key_cryptography&oldid=1148372231, 2023 [Online]. Available: https://en.wikipedia.org/wiki/Public-key_cryptography. [Accessed: Mar. 11, 2023]
- [63] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and Dr. C. Adams, “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP.” RFC 6960, Jun. 2013 [Online]. Available: <https://www.rfc-editor.org/info/rfc6960>
- [64] S. Helme, “Revocation is broken.” Jul. 03, 2017 [Online]. Available: <https://scotthelme.co.uk/revocation-is-broken/>. [Accessed: Apr. 20, 2023]
- [65] R. Sanders, “What is a certificate revocation list (CRL) vs OCSP?” Keyfactor, Nov. 27, 2020 [Online]. Available: <https://www.keyfactor.com/blog/what-is-a-certificate-revocation-list-crl-vs-ocsp/>. [Accessed: Apr. 20, 2023]
- [66] A. Gable, “A new life for certificate revocation lists.” Let’s Encrypt company blog; Let’s Encrypt, Sep. 07, 2022 [Online]. Available: <https://letsencrypt.org/2022/09/07/new-life-for-crls.html>. [Accessed: Apr. 20, 2023]
- [67] Y. N. Pettersen, “The Transport Layer Security (TLS) Multiple Certificate Status Request Extension.” RFC 6961, Jun. 2013 [Online]. Available: <https://www.rfc-editor.org/info/rfc6961>
- [68] P. Mockapetris, “Domain names - concepts and facilities.” RFC 1034, Nov. 1987 [Online]. Available: <https://www.rfc-editor.org/info/rfc1034>
- [69] P. Mockapetris, “Domain names - implementation and specification.” RFC 1035, Nov. 1987 [Online]. Available: <https://www.rfc-editor.org/info/rfc1035>
- [70] R. Elz and R. Bush, “Clarifications to the DNS Specification.” RFC 2181, Jul. 1997 [Online]. Available: <https://www.rfc-editor.org/info/rfc2181>
- [71] P. E. Hoffman, A. Sullivan, and K. Fujiwara, “DNS Terminology.” RFC 7719, Dec. 2015 [Online]. Available: <https://www.rfc-editor.org/info/rfc7719>
- [72] Microsoft Learn contributors, “Overview of DNS zones and records.” Microsoft Learn article via GitHub (<https://github.com/MicrosoftDocs/azure-docs/blob/main/articles/dns/dns-zones-records.md>); Microsoft Learn, Oct. 14, 2016 [Online]. Available: <https://docs.microsoft.com/en-us/azure/dns/dns-zones-records>. [Accessed: Apr. 21, 2023]
- [73] IANA, “Technical requirements for authoritative name servers.” IANA, 2013 [Online]. Available: <https://www.iana.org/help/nameserver-requirements>. [Accessed: Apr. 21, 2023]

- [74] G. Jevtic, “DNS best practices for security and performance.” PhoenixNAP, Nov. 11, 2019 [Online]. Available: <https://phoenixnap.com/kb/dns-best-practices-security>. [Accessed: Apr. 21, 2023]
- [75] Infoblox, “What is DNS security?” Infoblox company blog, 2020 [Online]. Available: <https://www.infoblox.com/dns-security-resource-center/dns-security-overview/>. [Accessed: Apr. 13, 2023]
- [76] L. Gyongyösi, “What is a DNS zone and how to keep safe from DNS zone transfer attacks.” Heimdal Security company blog; Heimdal Security, Jan. 20, 2023 [Online]. Available: <https://heimdalsecurity.com/blog/dns-zone/>. [Accessed: Apr. 13, 2023]
- [77] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, “Resource Records for the DNS Security Extensions.” RFC 4034, Mar. 2005 [Online]. Available: <https://www.rfc-editor.org/info/rfc4034>
- [78] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, “Protocol Modifications for the DNS Security Extensions.” RFC 4035, Mar. 2005 [Online]. Available: <https://www.rfc-editor.org/info/rfc4035>
- [79] D. Barr, “Common DNS Operational and Configuration Errors.” RFC 1912, Feb. 1996 [Online]. Available: <https://www.rfc-editor.org/info/rfc1912>
- [80] V. Dukhovni and W. Hardaker, “The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance.” RFC 7671, Oct. 2015 [Online]. Available: <https://www.rfc-editor.org/info/rfc7671>
- [81] pete, “Where does my ds record originate from?” Server Fault answer; Server Fault, Nov. 05, 2015 [Online]. Available: <https://serverfault.com/a/734034>. [Accessed: Apr. 21, 2023]
- [82] ICANN, “DNSSEC – what is it and why is it important?” ICANN, Mar. 05, 2019 [Online]. Available: <https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en>. [Accessed: Apr. 15, 2023]
- [83] Ó. Guðmundsson, “Adding Acronyms to Simplify Conversations about DNS-Based Authentication of Named Entities (DANE).” RFC 7218, Apr. 2014 [Online]. Available: <https://www.rfc-editor.org/info/rfc7218>
- [84] D. Crocker, “Scoped Interpretation of DNS Resource Records through ”Underscored” Naming of Attribute Leaves.” RFC 8552, Mar. 2019 [Online]. Available: <https://www.rfc-editor.org/info/rfc8552>
- [85] “Openssl-s_client.” OpenSSL man page; OpenSSL, 2023 [Online]. Available: https://www.openssl.org/docs/manmaster/man1/openssl-s_client.html. [Accessed: Mar. 01, 2023]
- [86] R. Barnes, M. Thomson, and E. Rescorla, “Unknown Key-Share Attacks on DNS-based Authentications of Named Entities (DANE),” Internet Engineering Task Force; Internet Engineering Task Force, Internet-Draft draft-barnes-dane-uks-00, Oct. 2016 [Online]. Available: <https://datatracker.ietf.org/doc/draft-barnes-dane-uks/00/>

- [87] P. Wouters, H. Tschofenig, J. I. Gilmore, S. Weiler, and T. Kivinen, “Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS).” RFC 7250, Jun. 2014 [Online]. Available: <https://www.rfc-editor.org/info/rfc7250>
- [88] V. Dukhovni, “Please avoid ‘3 0 1’ and ‘3 0 2’ DANE TLSA records with LE certificates.” Let’s Encrypt forum post; Let’s Encrypt, Dec. 15, 2015 [Online]. Available: <https://community.letsencrypt.org/t/please-avoid-3-0-1-and-3-0-2-dane-tlsa-records-with-le-certificates/7022>. [Accessed: Apr. 23, 2023]
- [89] J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels, “DNS Transport over TCP - Implementation Requirements.” RFC 7766, Mar. 2016 [Online]. Available: <https://www.rfc-editor.org/info/rfc7766>
- [90] P. E. Hoffman and P. McManus, “DNS Queries over HTTPS (DoH).” RFC 8484, Oct. 2018 [Online]. Available: <https://www.rfc-editor.org/info/rfc8484>
- [91] D. E. E. 3rd, Ó. Guðmundsson, P. A. Vixie, and B. Wellington, “Secret Key Transaction Authentication for DNS (TSIG).” RFC 2845, May 2000 [Online]. Available: <https://www.rfc-editor.org/info/rfc2845>
- [92] M. Dempsky, “DNSCurve: Link-Level Security for the Domain Name System,” Internet Engineering Task Force; Internet Engineering Task Force, Internet-Draft draft-dempsky-dnscurve-01, Feb. 2010 [Online]. Available: <https://datatracker.ietf.org/doc/draft-dempsky-dnscurve/01/>
- [93] V. Dukhovni and W. Hardaker, “SMTP Security via Opportunistic DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS).” RFC 7672, Oct. 2015 [Online]. Available: <https://www.rfc-editor.org/info/rfc7672>
- [94] A. Melnikov, “Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols.” RFC 7817, Mar. 2016 [Online]. Available: <https://www.rfc-editor.org/info/rfc7817>
- [95] M. Kucherawy and E. Zwicky, “Domain-based Message Authentication, Reporting, and Conformance (DMARC).” RFC 7489, Mar. 2015 [Online]. Available: <https://www.rfc-editor.org/info/rfc7489>
- [96] J. Hodges, C. Jackson, and A. Barth, “HTTP Strict Transport Security (HSTS).” RFC 6797, Nov. 2012 [Online]. Available: <https://www.rfc-editor.org/info/rfc6797>
- [97] C. Evans, C. Palmer, and R. Sleevi, “Public Key Pinning Extension for HTTP.” RFC 7469, Apr. 2015 [Online]. Available: <https://www.rfc-editor.org/info/rfc7469>
- [98] P. Hallam-Baker and R. Stradling, “DNS Certification Authority Authorization (CAA) Resource Record.” RFC 6844, Jan. 2013 [Online]. Available: <https://www.rfc-editor.org/info/rfc6844>
- [99] B. M. Schwartz, M. Bishop, and E. Nygren, “Service binding and parameter specification via the DNS (DNS SVCB and HTTPS RRs),” Internet Engineering Task Force; Internet Engineering Task Force, Internet-Draft draft-ietf-dnsop-svcb-https-12, Mar. 2023 [Online]. Available: <https://datatracker.ietf.org/doc/draft-ietf-dnsop-svcb-https/12/>

- [100] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, “Automatic Certificate Management Environment (ACME).” RFC 8555, Mar. 2019 [Online]. Available: <https://www.rfc-editor.org/info/rfc8555>
- [101] “IT Security and Privacy – A framework for identity management – Part 1: Terminology and concepts,” International Organization for Standardization, Geneva, CH, Standard, May 2019 [Online]. Available: <https://www.iso.org/standard/77582.html>
- [102] Wikipedia contributors, “Identity management — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Identity_management&oldid=1151190645, 2023 [Online]. Available: https://en.wikipedia.org/wiki/Identity_management. [Accessed: Mar. 30, 2023]
- [103] P. Saint-Andre and J. Hodges, “Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS).” RFC 6125, Mar. 2011 [Online]. Available: <https://www.rfc-editor.org/info/rfc6125>
- [104] P. A. Grassi, M. E. Garcia, and J. L. Fenton, “Digital identity guidelines,” National Institute of Standards; Technology, Standard, Jun. 2017 [Online]. Available: <https://pages.nist.gov/800-63-3/sp800-63-3.html>
- [105] Let’s Encrypt, “Let’s encrypt FAQ – what is the lifetime for let’s encrypt certificates? For how long are they valid?” Let’s Encrypt, 2017 [Online]. Available: <https://letsencrypt.org/docs/faq/#what-is-the-lifetime-for-let-s-encrypt-certificates-for-how-long-are-they-valid>. [Accessed: Apr. 25, 2023]
- [106] Internet Systems Consortium, “host - DNS lookup utility.” 2023 [Online]. Available: <https://bind9.readthedocs.io/en/latest/manpages.html#host-dns-lookup-utility>. [Accessed: Apr. 17, 2023]
- [107] Internet Systems Consortium, “dig - DNS lookup utility.” 2023 [Online]. Available: <https://bind9.readthedocs.io/en/latest/manpages.html#dig-dns-lookup-utility>. [Accessed: Apr. 17, 2023]
- [108] “Nmap: The network mapper - free security scanner.” 2023 [Online]. Available: <https://nmap.org/>
- [109] J. Quinteiro, “Displaying a remote SSL certificate details using CLI tools.” Server Fault Q&A; Server Fault, Nov. 01, 2017 [Online]. Available: <https://serverfault.com/a/881415>. [Accessed: Apr. 24, 2023]
- [110] “Openssl.” OpenSSL man page; OpenSSL, 2023 [Online]. Available: <https://www.openssl.org/docs/manmaster/man1/openssl.html>
- [111] “Certbot.” 2023 [Online]. Available: <https://certbot.eff.org/>

Appendix

Code samples

A small example using TLS certificate PKI with OpenSSL, written in Bash.

```
#!/bin/bash

mkdir -- 'CA' 'Alice' 'Bob'

# Step 1: Make certificates

(
  cd -- 'CA'
  openssl 'genpkey' -algorithm 'RSA' -pkeyopt 'rsa_keygen_bits:2048' \
    -out 'private.pem'
  openssl 'pkey' -in 'private.pem' -pubout -out 'public.pem'
  openssl 'req' -x509 -noenc -key 'private.pem' -sha256 -days '1024' \
    -out 'root.pem'
)

(
  cd -- 'Alice'
  openssl 'genpkey' -algorithm 'RSA' -pkeyopt 'rsa_keygen_bits:2048' \
    -out 'private.pem'
  openssl 'pkey' -in 'private.pem' -pubout -out 'public.pem'
  openssl 'req' -new -key 'private.pem' -out 'request.csr'
)

(
  cd -- 'Bob'
  openssl 'genpkey' -algorithm 'RSA' -pkeyopt 'rsa_keygen_bits:2048' \
    -out 'private.pem'
  openssl 'pkey' -in 'private.pem' -pubout -out 'public.pem'
  openssl 'req' -new -key 'private.pem' -out 'request.csr'
)
```

```

openssl 'x509' -req -in 'Alice/request.csr' -CA 'CA/root.pem' \
    -CAkey 'CA/private.pem' -CAcreateserial -out 'Alice/certificate.pem' \
    -days '512' -sha256
openssl 'x509' -req -in 'Bob/request.csr' -CA 'CA/root.pem' \
    -CAkey 'CA/private.pem' -CAcreateserial -out 'Bob/certificate.pem' \
    -days '512' -sha256

# Step 2: Verify certificates, get SPKI

(
    cd -- 'Alice'
    openssl 'verify' -CAfile '../CA/root.pem' '../Bob/certificate.pem'
    openssl 'x509' -pubkey -in '../Bob/certificate.pem' -noout > 'public-Bob.pem'
)

(
    cd -- 'Bob'
    openssl 'verify' -CAfile '../CA/root.pem' '../Alice/certificate.pem'
    openssl 'x509' -pubkey -in '../Alice/certificate.pem' \
        -noout > 'public-Alice.pem'
)

# Step 3: Alice picks a symmetric key, encrypts it using Bob's public key,
#         hashes it and creates a signature using her private key

(
    cd -- 'Alice'
    openssl 'rand' -base64 -out 'symmetric.pem' '32'
    openssl 'pkeyutl' -encrypt -in 'symmetric.pem' -pubin \
        -inkey 'public-Bob.pem' -out 'symmetric.enc'
    openssl 'dgst' -sha1 -sign 'private.pem' -out 'signature.bin' 'symmetric.pem'
)

# Step 4: Bob receives encrypted symmetric key and signature from Alice

(
    cd -- 'Bob'
    openssl 'pkeyutl' -decrypt -in '../Alice/symmetric.enc' \
        -inkey 'private.pem' -out 'symmetric.pem'
    openssl 'dgst' -sha1 -verify 'public-Alice.pem' \
        -signature '../Alice/signature.bin' 'symmetric.pem'
)

```

```
# Step 5: Alice encrypts data using the symmetric key,
#         Bob receives and decrypts it

(
  cd -- 'Alice'
  openssl 'enc' -aes-256-cbc -pbkdf2 -pass 'file:symmetric.pem' -p -md \
    'sha256' -in 'plaintext.txt' -out 'ciphertext.bin'
)

(
  cd -- 'Bob'
  openssl 'enc' -aes-256-cbc -pbkdf2 -d -pass 'file:symmetric.pem' -p \
    -md 'sha256' -in '../Alice/ciphertext.bin' -out 'plaintext.txt'
)
```

























Design wireframes

DANE Identity Manager

DNS management

Identity management

Zone: example.com.

| Name | TTL | Class | Type | Value | Actions |
|-----------------|------|-------|------|-----------|---|
| lorem | 3600 | IN | A | 192.0.2.1 |   |
| ipsum | 3600 | IN | A | 192.0.2.2 |   |
| dolor | 3600 | IN | A | 192.0.2.3 |   |
| sit | 3600 | IN | A | 192.0.2.4 |   |
| amet | 3600 | IN | A | 192.0.2.5 |   |
| bar | 3600 | IN | A | 192.0.2.6 |   |
| _443._tcp.lorem | 3600 | IN | TLSA | 3 1 1 ... |   |
| _443._tcp.ipsum | 3600 | IN | TLSA | 3 1 1 ... |   |
| _443._tcp.dolor | 3600 | IN | TLSA | 2 0 1 ... |   |
| _443._tcp.sit | 3600 | IN | TLSA | 2 0 1 ... |   |
| _443._tcp.amet | 3600 | IN | TLSA | 1 1 1 ... |   |
| _443._tcp.amet | 3600 | IN | TLSA | 3 1 1 ... |   |


+ Add











DANE Identity Manager

DNS management

Identity management

TLS | IPsec | SSH | OpenPGP | S/MIME




 Settings

| Label | Hostnames | Valid from | Valid until | Type | Status | Actions |
|------------|---|------------|-------------|-------------------------------|----------|---|
| loremipsum | lorem.example.com, ipsum.example.com | 2023-04-01 | 2023-07-01 | DANE-EE SPKI | Deployed |   |
| dolorsit | dolor.example.com, sit.example.com | 2023-04-01 | 2024-04-01 | DANE-TA Cert | Deployed |   |
| amet | amet.example.com | 2023-04-01 | 2024-04-01 | PKIX-EE SPKI, DANE-EE SPKI | Deployed |   |
| foo | foo.example.com | 2023-04-01 | 2023-07-01 | (none) | No DANE |   |
| bar | bar.example.com | 2023-04-01 | 2023-07-01 | | Revoked |   |

+ Add

Clicking on the pencil icon in TLSA RR rows in *DNS management* or on the pencil icon in *Identity management* will bring up the dialog *TLS identity (DANE TLSA)*. Clicking on the *Settings* button will bring up the *Global DANE TLSA settings* dialog.

TLS identity (DANE TLSA)

TLS certificate:  Upload  Delete  Replace

-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----





Name: loremipsum



Applies to: lorem.example.com
ipsum.example.com


Valid from: 2023-04-01

Valid until: 2023-07-01

Automatically renews on: 2023-06-01

DANE TLSA associations: DANE-EE  SPKI   Remove
 Add

Certificate signing request:  

CA auth token: 

TLS: active

DANE: active

View TLSA records

Global DANE TLSA settings

Enabled hashing algorithms:

☒ SHA2-256

☐ SHA2-512

72