Transport Layer Security
Internet-Draft
Intended status: Standards Track
Expires: 2 November 2026

B. Beck
OpenSSL
D. Benjamin
Google LLC
D. O'Brien

K. Nekritz
Meta

1 May 2026

TLS Trust Anchor Identifiers
draft-ietf-tls-trust-anchor-ids-04

Abstract

This document defines the TLS Trust Anchors extension, a mechanism for relying parties to convey trusted certification authorities. It describes individual certification authorities more succinctly than the TLS Certificate Authorities extension.

Additionally, to support TLS clients with many trusted certification authorities, it supports a mode where servers describe their available certification paths and the client selects from them. Servers may describe this during connection setup, or in DNS for lower latency.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://tlswg.github.io/tls-trust-anchor-ids/draft-ietf-tls-trust-anchor-ids.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-tls-trust-anchor-ids/.

Discussion of this document takes place on the Transport Layer Security Working Group mailing list (mailto:tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/. Subscribe at https://www.ietf.org/mailman/listinfo/tls/.

Source for this draft and an issue tracker can be found at https://github.com/tlswg/tls-trust-anchor-ids.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This Internet-Draft will expire on 2 November 2026.

Copyright Notice

Table of Contents

1. Introduction

TLS [RFC8446] authentication uses X.509 certificates [RFC5280] to associate the _authenticating party's_ TLS key with its application identifiers, such as DNS names. These associations are signed by some certification authority (CA). The peer, or _relying party_, curates a set of CAs that are trusted to only sign correct associations, which allows it to rely on the TLS to authenticate application identifiers. Typically the authenticating party is the server and the relying party is the client.

An authenticating party may need to interoperate with relying parties that trust different sets of CAs. Section 4.2.4 of [RFC8446] defines the certificate_authorities extension to accommodate this. It allows the authenticating party to provision multiple certificates and select the one that will allow the relying party to accept its TLS key. This is analogous to parameter negotiation elsewhere in TLS.

However, certificate_authorities's size is impractical for some applications. Existing PKIs may have many CAs, and existing CAs may have long X.509 names. As of August 2023, the Mozilla CA Certificate Program [MOZILLA-ROOTS] contained 144 CAs, with an average name length of around 100 bytes. Such TLS deployments often do not use trust anchor negotiation at all.

Without a negotiation mechanism, the authenticating party must obtain a single certificate that simultaneously satisfies all relying parties. This is challenging when relying parties are diverse. PKI transitions, including those necessary for user security, naturally lead to relying party diversity, so the result is that service availability conflicts with security and overall PKI evolution:

* For an authenticating party to use a CA in its single certificate, all supported relying parties must trust the CA. PKI transitions then become difficult when authenticating parties support older, unupdated relying parties. This impacts both new keys from existing CA operators and new CA operators.

* When a relying party must update its policies to meet new security requirements, it adds to relying party diversity and the challenges that authenticating parties and CAs face. The relying party must then choose between compromising on user security or burdening the rest of the ecosystem, potentially impacting availability in the process.

To address this, this document introduces Trust Anchor Identifiers (Trust Anchor IDs). There are several parts to this mechanism:

1. Section 3 defines _trust anchor IDs_, which are short, unique identifiers for X.509 trust anchors, or groups of trust anchors.

2. Section 4 defines a TLS extension that communicates the relying party's requested trust anchors using trust anchor IDs. IDs that represent individual trust anchors can mitigate long X.509 names. IDs that represent groups of trust anchors can mitigate large trust anchor lists.

3. Section 4.3 defines a retry mechanism that, when the relying party is a TLS client, can mitigate signaling failures. The server provides its available trust anchors alongside its certificate, so that the client can retry on mismatch. This can further mitigate large trust anchor lists by allowing the client to initially omit some trust anchors or use an otherwise too broad trust anchor group. However, this mitigation can come at the cost of additional round trips in some cases.

4. Section 6, finally, allows TLS servers to advertise their available trust anchors in HTTPS or SVCB [RFC9460] DNS records. When the above options are insufficient, TLS clients can request an accurate initial subset and avoid a retry penalty.

Together, they reduce the size costs of trust anchor negotiation, supporting flexible and robust PKIs for more applications.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document additionally uses the TLS presentation language, defined in Section 3 of [RFC8446], and ASN.1, defined in [X680].

2.1. Terminology and Roles

This document discusses three roles:

Authenticating party: The party authenticating itself in the protocol. In TLS, this is the side sending the Certificate and CertificateVerify message.

Relying party: The party whom the authenticating party presents its identity to. In TLS, this is the side that validates a Certificate and CertificateVerify message.

Certification authority (CA): The service issuing certificates to the authenticating party.

Additionally, there are several terms used throughout this document to describe this proposal:

Trust anchor: A pre-distributed X.509 name and public key that

relying parties use to determine whether a certification path is trusted. See Section 6.1.1 of [RFC5280]. Trust anchors are sometimes configured as self-signed certificates.

Certification path: An ordered list of X.509 certificates starting with the target certificate. Each certificate is issued by the next certificate, except the last, which is issued by a trust anchor.

3. Trust Anchor Identifiers

This section defines trust anchor IDs, which are short, unique identifiers that represent either a trust anchor, or a group of trust anchors. When a trust anchor ID represents a group of trust anchors, it is known as a _trust anchor group_.

A trust anchor ID is an object identifier (OID) [X680] under the OID arc of some IANA-registered Private Enterprise Number (PEN) [RFC9371]. For compactness, they are represented as relative object identifiers (see Section 33 of [X680]), relative to the OID prefix 1.3.6.1.4.1. For example, an organization with PEN 32473 might define a trust anchor ID with the OID 1.3.6.1.4.1.32473.1. As a relative object identifier, it would be the OID 32473.1.

Depending on the protocol, trust anchor IDs may be represented in one of three ways:

* For use in ASN.1-based protocols, a trust anchor ID’s ASN.1 representation is the relative object identifier described above. This may be encoded in DER [X690], or some other ASN.1 encoding. The example ID’s DER encoding is the six-octet sequence {0x0d, 0x04, 0x81, 0xfd, 0x59, 0x01}.

* For use in binary protocols such as TLS, a trust anchor ID’s binary representation consists of the contents octets of the relative object identifier’s DER encoding, as described in Section 8.20 of [X690]. Note this omits the tag and length portion of the encoding. The example ID’s binary representation is the four-octet sequence {0x81, 0xfd, 0x59, 0x01}.

* For use in ASCII-compatible text protocols, a trust anchor ID’s ASCII representation is the relative object identifier in dotted decimal notation. The example ID’s ASCII representation is 32473.1.

The length of a trust anchor ID’s binary representation MUST NOT exceed 255 bytes. It SHOULD be significantly shorter, for bandwidth efficiency.

A trust anchor ID representing a single trust anchor SHOULD be allocated by the CA operator and be common among relying parties that trust the CA. They MAY be allocated by another party, e.g. when bootstrapping an existing ecosystem, if all parties agree on the ID. In particular, the protocol requires authenticating and relying parties to agree, and the authenticating party's configuration typically comes from the CA.

A trust anchor ID representing a trust anchor group MAY be allocated by any party. However, to be useful, the group requires agreement between relying parties and authenticating parties. Section 5 discusses defining trust anchor groups in more detail.

3.1. Trust Anchor Ranges

Related trust anchor IDs can be allocated from a single OID arc, such as in the versioning construction described in Section 5.1. This section defines a _trust anchor range_, which describes a series of such IDs. Concretely, a trust anchor range is defined by three properties:

* A trust anchor ID base

* Two non-negative, 64-bit integers min and max

A trust anchor range is said to _contain_ some trust anchor ID, id if the id, as a relative OID, is the concatenation of base and some integer component between min and max, inclusive. max can be set to 2^64-1 if there is no upper bound.

The following procedure can be used to perform this check. It succeeds if the range contains id and fails otherwise:

1. Check that base does not end in the middle of an OID component. That is, check that the most-significant bit of the last byte of base is unset. If it is set, fail the procedure.

2. Check that base is a prefix of id. If not, fail the procedure. Let rest be id with the base prefix removed.

3. Decode rest as a minimally-encoded, big-endian, base-128 OID component as follows:

    1. If rest is empty, fail the procedure.

    2. If the most-significant bit of the last byte of rest is set, fail the procedure.

3. If the most-significant bit of any other byte of rest is unset, fail the procedure.

4. If the first byte of rest is 0x80, fail the procedure.

5. Set v to zero. Throughout this procedure, v will be less than 2^64.

6. For each byte b of rest:

    1. If v is greater than or equal to 2^57, fail the procedure.

    2. Set v to (v << 7) + (b & 127).

4. Check if min <= v <= max. If this is not true, fail the procedure. Otherwise, the procedure succeeds.

3.2. Authenticating Party Configuration

Authenticating parties are configured with one or more candidate certification paths to present in TLS, in some preference order. This preference order is used when multiple candidate paths are usable for a connection. For example, the authenticating party may prefer candidates that minimize message size or have more performant private keys.

Each candidate path that participates in this protocol must be configured with two properties:

* The trust anchor ID for its corresponding trust anchor.

* Optionally, a list of _trust anchor group inclusions_. A trust anchor group inclusion is a trust anchor range (Section 3.1) that describes some trust anchor groups also containing the path's trust anchor.

These properties allow certificate selection (see Section 4.2) to consider this path when a relying party advertises a matching trust anchor ID. It is RECOMMENDED, though not required, that this information come from the CA. Section 7 defines a RECOMMENDED format for this information, along with an optional ACME [RFC8555] extension for CAs to send it.

Authenticating parties MAY have candidate certification paths without these associated properties. Such paths will not participate in this protocol. They MAY participate in other trust anchor negotiation protocols, such as the certificate_authorities extension.

3.3. Relying Party Configuration

Relying parties are configured with one or more supported trust anchors. Each trust anchor that participates in this protocol must have an associated trust anchor ID.

When trust anchors are represented as X.509 certificates, the X.509 trust anchor ID extension MAY be used to carry this ID. The trust anchor ID extension has an extnID of id-pe-trustAnchorID and an extnValue containing a DER-encoded TrustAnchorID structure, defined below. The TrustAnchorID is the trust anchor ID's ASN.1 representation, described in Section 3. This extension MUST be non-critical.

```
id-pe-trustAnchorID OBJECT IDENTIFIER ::=
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-pe(1) TBD }

TrustAnchorID ::= RELATIVE-OID
```

Relying parties MAY instead or additionally configure trust anchor IDs via some application-specific out-of-band information.

Relying parties MAY additionally be configured with trust anchor groups that include their trust anchors. When authenticating parties are known to be configured with corresponding inclusion lists (Section 3.2), this can further reduce the size of messages sent by the relying party.

Relying parties MAY support trust anchors without associated trust anchor IDs, but such trust anchors will not participate in this protocol. Those trust anchors MAY participate in other trust anchor negotiation protocols, such as the certificate_authorities extension.

4. TLS Extension

This section defines the trust_anchors extension, which is sent in the ClientHello, EncryptedExtensions, CertificateRequest, and Certificate messages in TLS 1.3 or later.

4.1. Overview

The trust_anchors extension is defined using the structures below:

```
enum { trust_anchors(TBD), (2^16-1) } ExtensionType;

opaque TrustAnchorID<1..2^8-1>;

/* These types differ in whether they may be empty. */
TrustAnchorID RequestedTrustAnchorList<0..2^16-1>;
TrustAnchorID AvailableTrustAnchorList<1..2^16-1>;
```

When the extension is sent in the ClientHello or CertificateRequest messages, the extension_data is a RequestedTrustAnchorList and indicates that the sender supports the specified trust anchors or trust anchor groups. The list is unordered, and MAY be empty. Each TrustAnchorID uses the binary representation, as described in Section 3.

When the extension is sent in EncryptedExtensions, the extension_data is an AvailableTrustAnchorList containing the list of individual trust anchors for which the server has a candidate path. This list is ordered matching the server's candidate path preference order, and MUST NOT be empty.

When the extension is sent in Certificate, the extension_data MUST be empty and indicates that the sender sent the certificate because the certificate matched a trust anchor ID sent by the peer. When used in this form, the extension may only be sent in the first CertificateEntry. It MUST NOT be sent in subsequent ones.

4.2. Certificate Selection

A trust_anchors extension in the ClientHello or CertificateRequest is processed similarly to the certificate_authorities extension. The relying party indicates some set of supported trust anchors in the ClientHello or CertificateRequest trust_anchors extension. The authenticating party then selects a certificate from its candidate certification paths (see Section 3.2), as described in Section 4.4.2.2 of [RFC8446] and Section 4.4.2.3 of [RFC8446]. This process is extended as follows:

If the ClientHello or CertificateRequest contains a trust_anchors extension, the authenticating party SHOULD send a certification path such that either:

* The path's trust anchor ID appears in the relying party's trust_anchors extension.

* One of the path's trust anchor group inclusions contains some ID in the relying party's trust_anchors extension.

If the ClientHello or CertificateRequest contains both trust_anchors and certificate_authorities, certification paths that satisfy either extension's criteria may be used. This additionally applies to future extensions which play a similar role.

If no certification paths satisfy either extension, the authenticating party MAY return a handshake_failure alert, or choose among fallback certification paths without considering trust_anchors or certification_authorities. See Section 4.3 for additional guidance on selecting a fallback when the ClientHello contains trust_anchors.

Sending a fallback allows the authenticating party to retain support for relying parties that do not implement any form of trust anchor negotiation. In this case, the authenticating party must find a sufficiently ubiquitous trust anchor, if one exists. However, only those relying parties need to be considered in this ubiquity determination. Updated relying parties may continue to evolve without restricting fallback certificate selection.

If the authenticating party sends a certification path that matches the relying party's trust_anchors extension, as described in Section 4.2, the authenticating party MUST send an empty trust_anchors extension in the first CertificateEntry of the Certificate message. In this case, the certificate_list flexibility described in Section 4.4.2 of [RFC8446] no longer applies. The certificate_list MUST contain a complete certification path, issued by the matching trust anchor, correctly ordered and with no extraneous certificates. That is, each certificate MUST certify the one immediately preceding it, and the trust anchor MUST certify the final certificate. The authenticating party MUST NOT send the trust_anchors extension in the Certificate message in other situations.

If a relying party receives this extension in the Certificate message, it MAY choose to disable path building [RFC4158] and validate the peer's certificate list as pre-built certification path. Doing so avoids the unpredictable behavior of path-building, and helps ensure CAs and authenticating parties do not inadvertently provision incorrect paths.

4.3. Retry Mechanism

When the relying party is a client, it may choose not to send its full trust anchor ID list due to fingerprinting risks (see Section 9), or because the list is too large. The client MAY send a subset of supported trust anchors, or an empty list. This subset may be determined by, possibly outdated, prior knowledge about the server, such as Section 6 or past connections.

To accommodate this, when receiving a ClientHello with trust_anchors, the server collects all candidate certification paths which:

* Have a trust anchor ID, and

* Satisfy the conditions in Section 4.4.2.2 of [RFC8446], with the exception of certification_authorities, and any future extensions that play a similar role

If this collection is non-empty, the server sends a trust_anchors extension in EncryptedExtensions, containing the corresponding trust anchor IDs in preference order.

When a client sends a subset or empty list in trust_anchors, it SHOULD implement the following retry mechanism:

If the client receives either a connection error or an untrusted certificate, the client looks in server's EncryptedExtensions for a trust anchor ID that it trusts. If there are multiple, it selects an option based on the server's preference order and its local preferences. It then makes a new connection to the same endpoint, sending only the selected trust anchor ID in the ClientHello trust_anchors extension. If the EncryptedExtensions had no trust_anchor extension, or no match was found, the client returns the error to the application.

Clients SHOULD retry at most once per connection attempt.

[[TODO: Retrying in a new connection is expensive and cannot be done from within the TLS stack in most implementations. Consider handshake modifications to instead retry within the same connection. https://github.com/tlswg/tls-trust-anchor-ids/issues/53]]

This mechanism allows the connection to recover from a certificate selection failure, e.g. due to the client not revealing its full preference list, at additional latency cost. Section 6 describes an optimization which can avoid this cost.

This mechanism also allows servers to safely send fallback certificates that may not be as ubiquitously acceptable. Without some form of trust anchor negotiation, servers are limited to selecting certification paths that are ubiquitously trusted in all supported clients. This often means sending extra cross-certificates to target the lowest common denominator at a bandwidth cost. If the ClientHello contains trust_anchors, the server MAY opportunistically send a less ubiquitous, more bandwidth-efficient path based on local heuristics, with the expectation that the client will retry when the heuristics fail.

5. Trust Anchor Groups

A trust anchor ID is typically much smaller than the corresponding X.509 name. Depending on the number of trust anchors, this can be sufficient to efficiently represent relying party state.

PKIs where further size savings are needed can use trust anchor groups (Section 3). Trust anchor groups require additional coordination within a PKI, but they can further reduce relying party message sizes by allowing one ID to signal multiple trust anchors. To be usable, a trust anchor group must:

* be known to and sent by relying parties (see Section 3.3); and

* configured with candidate paths in authenticating parties (see Section 3.2), ideally provided by the CA during issuance (see Section 7).

This document does not prescribe how to define trust anchor groups, but gives some general guidance:

A trust anchor group specifies a collection of trust anchors, which a relying party can send to represent the contents. For example:

* A set of root CAs (or intermediate CAs, as in Section 8.5) operated by a CA operator.

* A set of trust anchors common to large set of relying parties.

* A set of related application-specific trust anchors, such as a range of Merkle Tree Certificate landmarks [I-D.ietf-plants-merkle-tree-certs].

Different group definitions trade off size savings, applicability, and coordination overhead. A group that reflects a single CA operator will cover fewer trust anchors, so a relying party might combine several operators' IDs to describe its trust anchors.

However, it is generally usable by relying parties that trust this CA operator. Such a group also requires minimal coordination for the CA operator to provide group inclusion information (Section 3.2) with the certificate.

Conversely, a group that reflects a single relying party vendor can potentially be the only ID sent. However, it may be less generally usable when relying parties differ. Groups reflecting multiple relying party vendors are more broadly usable, but may need to be combined with other IDs in a given relying party. For example, a relying party might send a group containing established CAs common to its ecosystem, and individual IDs for its remaining, not yet as common CAs.

A client relying party MAY send a group containing CAs it does not trust, however it SHOULD then be prepared to retry (see Section 4.3) in case of signaling failure.

The authenticating party selection process described in Section 4.2 can implemented generically for any trust anchor group. This allows deployments to tailor their group allocation based on their needs, without requiring software updates in authenticating parties. Where feasible, deployments SHOULD use groups that are more broadly applicable and require lower coordination overhead.

5.1. Versioned Groups

Over time, a group may become out-of-date, making it describe current relying parties less effectively. For example, a CA operator may deploy or turn down a CA instance, or a relying party may trust a new CA or distrust an existing CA. Existing trust anchor groups SHOULD NOT be redefined, but the following versioning scheme MAY be used to define updated groups:

A versioned sequence of trust anchor groups is identified by a OID arc. Each group has an ID of this OID arc, with a non-negative integer version number component appended. For example, versioned groups using the OID arc 32473.2 would have IDs 32473.2.0, 32473.2.1, 32473.2.2, and so on. When defining a new group version, the version component is incremented.

The trust anchor group inclusion for a candidate path is a trust anchor range (Section 3.1) determined as follows:

* At issuance, if the trust anchor is no longer in the latest group version, the range’s min and max values are the first and last version that include the trust anchor, respectively.

* At issuance, if the trust anchor is in the latest group version, the range's min value is the first version that includes the trust anchor, and its max value is 2^64-1.

In the second case, the range contains not-yet-defined group versions, so there is a potential signaling error. Suppose, after issuance, a new group version is defined without the trust anchor. The unlimited upper bound is now incorrect. A relying party might not trust this trust anchor, while sending this new group version. However, the authenticating party will misinterpret the certificate as compatible based on its stale information. Such signaling errors may result in the wrong certificate being selected.

This can be mitigated in one several ways:

* Only pre-existing certificates are impacted. Newly-issued certificates postdate this version and will have the correct upper bound. When the certificate is renewed, group inclusions will be corrected.

* [SCTNotAfter] describes a trust anchor removal strategy that only impacts newly-issued certificates. In this case, no renewal is needed. Pre-existing group inclusions remain accurate under this strategy.

* If the authenticating party's preferences place the correct candidate path (issued by a newer trust anchor) ahead of misinterpreted one (issued by the removed trust anchor), the correct candidate will still be chosen.

* When the relying party is a client, any remaining signaling errors can be corrected with the retry mechanism described in Section 4.3.

6. DNS Service Parameter

This section defines the tls-trust-anchors SvcParamKey [RFC9460]. TLS servers can use this to advertise their available trust anchors in DNS, and aid the client in formulating its trust_anchors extension (see Section 4.3). This allows TLS deployments to support clients with many trust anchors without incurring the overhead of a reconnect.

6.1. Syntax

The tls-trust-anchors parameter contains an ordered list of one or more trust anchor IDs, in server preference order.

The presentation value of the SvcParamValue is a non-empty comma-separated list (Appendix A.1 of [RFC9460]). Each element of the list is a trust anchor ID in the ASCII representation defined in Section 3. Any other value is a syntax error. To enable simpler parsing, this SvcParam MUST NOT contain escape sequences.

The wire format of the SvcParamValue is determined by prefixing each trust anchor ID with its length as a single octet, then concatenating each of these length-value pairs to form the SvcParamValue. These pairs MUST exactly fill the SvcParamValue; otherwise, the SvcParamValue is malformed.

For example, if a TLS server has three available certification paths issued by 32473.1, 32473.2.1, and 32473.2.2, respectively, the DNS record in presentation syntax may be:

```
example.net.  7200  IN SVCB 3 server.example.net. (
    tls-trust-anchors=32473.1,32473.2.1,32473.2.2 )
```

The wire format of the SvcParamValue would be the 17 octets below. In the example, the octets comprising each trust anchor ID are placed on separate lines for clarity

```
0x04, 0x81, 0xfd, 0x59, 0x01,
0x05, 0x81, 0xfd, 0x59, 0x02, 0x01,
0x05, 0x81, 0xfd, 0x59, 0x02, 0x02,
```

6.2. Configuring Services

Services SHOULD include the trust anchor ID for each of their available certification paths, in preference order, in the tls-trust-anchors of their HTTPS or SVCB endpoints. As TLS configuration is updated, services SHOULD update the DNS record to match. The mechanism for this is out of scope for this document, but services are RECOMMENDED to automate this process.

Services MAY have certification paths without trust anchor IDs, but those paths will not participate in this mechanism.

6.3. Client Behavior

When connecting to a service endpoint whose HTTPS or SVCB record contains the tls-trust-anchors parameter, the client first computes the intersection between its configured trust anchors and the server’s provided list. If this intersection is non-empty, the client MAY use it to determine the trust_anchors extension in the ClientHello (see Section 4.3).

If doing so, the client MAY send a subset of this intersection to meet size constraints, but SHOULD offer multiple options. This reduces the chance of a reconnection if, for example, the first option in the intersection uses a signature algorithm that the client doesn't support, or if the TLS server and DNS configuration are out of sync.

Although this service parameter is intended to reduce trust anchor mismatches, mismatches may still occur in some scenarios. Clients and servers MUST continue to implement the provisions described in Section 4.3, even when using this service parameter.

7. Certificate Properties

As described in Section 3.2, certification paths participating in this mechanism must be configured with a trust anchor ID. This section introduces a RECOMMENDED extensible CertificatePropertyList structure for representing this and other additional properties of a certification path. CertificatePropertyLists may be used as part of authenticating party configuration, and for CAs to communicate additional properties during certificate issuance.

The extensibility aims to simplify application deployment as PKI mechanisms evolve. When certificate issuance and application software is updated to pass this structure to the underlying TLS implementation, new properties may be transparently defined without changes to certificate and configuration management.

A CertificatePropertyList is defined using the TLS presentation language (Section 3 of [RFC8446]) below:

```
enum {
    trust_anchor_id(0),
    trust_anchor_group_inclusions(1),
    (2^16-1)
} CertificatePropertyType;

struct {
    CertificatePropertyType type;
    opaque data<0..2^16-1>;
} CertificateProperty;

CertificateProperty CertificatePropertyList<0..2^16-1>;
```

The entries in a CertificatePropertyList MUST be sorted numerically by type and MUST NOT contain values with a duplicate type. Inputs that do not satisfy these invariants are syntax errors and MUST be rejected by parsers.

This document defines two properties:

* trust_anchor_id, defined in Section 7.1

* trust_anchor_group_inclusions, defined in Section 7.2

Future documents MAY define other properties for use with other mechanisms. Such a document MUST define the format of the data field and how authenticating parties interpret the property. Authenticating parties MUST ignore properties with unrecognized CertificatePropertyType values.

7.1. Trust Anchor ID Property

The trust_anchor_id property's data field contains the binary representation of the trust anchor ID of the certification path's trust anchor, as described in Section 3.2.

7.2. Trust Anchor Group Inclusions Property

The trust_anchor_group_inclusions property's data field contains the certification path's trust anchor group inclusions, as described in Section 3.2. Each trust anchor group inclusion is described in a TrustAnchorRangeList structure, defined below. Each TrustAnchorRange structure describes a trust anchor range, as defined in Section 3.1.

```
struct {
    TrustAnchorID base;
    uint64 min;
    uint64 max;
} TrustAnchorRange;

TrustAnchorRange TrustAnchorRangeList<1..2^16-1>;
```

7.3. Media Type

A certification path with its associated CertificatePropertyList may be represented in a PEM [RFC7468] structure in a file of type "application/pem-certificate-chain-with-properties". Files of this type MUST use the strict encoding and MUST NOT include explanatory text. The ABNF [RFC5234] for this format is as follows, where "stricttextualmsg" and "eol" are as defined in Section 3 of [RFC7468]:

```
certchainwithproperties = stricttextualmsg eol stricttextualmsg
                          *(eol stricttextualmsg)
```

The first element MUST be the encoded CertificatePropertyList. The second element MUST be an end-entity certificate. Each following certificate MUST directly certify the one preceding it. The certificate representing the trust anchor MUST be omitted from the path.

CertificatePropertyLists are encoded using the "CERTIFICATE PROPERTIES" label. The encoded data is a serialized CertificatePropertyList, defined in Section 7.

Certificates are encoded as in Section 5.1 of [RFC7468], except DER [X690] MUST be used.

The following is an example file with a certification path containing an end-entity certificate and an intermediate certificate.

```
-----BEGIN CERTIFICATE PROPERTIES-----
TODO fill in an example
-----END CERTIFICATE PROPERTIES-----
-----BEGIN CERTIFICATE-----
TODO fill in an example
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
TODO fill in an example
-----END CERTIFICATE-----
```

The IANA registration for this media type is described in Section 11.2.

7.4. ACME Extension

The format defined in Section 7.3 can be used with ACME's alternate format mechanism (see Section 7.4.2 of [RFC8555]) as follows. When downloading certificates, a supporting client SHOULD include "application/pem-certificate-chain-with-properties" in its HTTP Accept header (Section 12.5.1 of [RFC9110]). When a supporting server sees such a header, it MAY then respond with that format to include a CertificatePropertyList with the certification path. This CertificatePropertyList MAY include trust_anchor_id and trust_anchor_group_inclusions properties for use with this protocol, or other properties defined in another document.

When used with ACME's alternate certificate chain mechanism (see Section 7.4.2 of [RFC8555]), this protocol removes the need for heuristics in determining which path to serve to which relying party.

The authenticating party MAY combine the resulting certification paths with those from other ACME orders, or other sources, for a complete set of candidate paths to serve.

8. Use Cases

trust_anchors, like certificate_authorities, implements trust anchor negotiation. That is, it allows an authenticating party to incorporate relying party trust anchors into certificate selection. trust_anchors allows a wider range of TLS applications to use trust anchor negotiation, notably those that would be unable to use certificate_authorities due to size or privacy limitations.

Without trust anchor negotiation, authenticating parties are limited to CAs in the intersection of all supported relying parties. However, trust anchors can vary significantly between different relying party implementations and different versions of a single relying party implementation, particularly as PKIs evolve to meet user security needs.

As security-positive PKI changes increase variance, this intersection shrinks. This leads to a conflict between user security and service availability. When the authenticating party cannot serve a certificate in the intersection, either the relying party must risk user security by not changing the PKI, or the authenticating party must degrade service availability by dropping support for some relying parties.

The rest of this section discusses uses cases for trust anchor negotiation.

8.1. Making Use of Newly-Trusted CAs

When one relying party trusts a new CA, other relying parties, such as older ones, may not yet trust it. Trust anchor negotiation allows an authenticating party to negotiate a certificate from the newer CA with relying parties that do trust it, while continuing to negotiate another certificate with relying parties that do not. This allows PKI transitions to progress smoothly. Connections can make use of, for example, a new CA’s stronger signature algorithms, stronger validation practices, better automation, or more efficient certificate sizes, without interruptions to other connections.

Without negotiation, the authenticating party is limited to its relying parties’ intersection and must wait for every supported relying party to be updated before the transition even begins. This wait could often take many years. In some cases, such as with IoT devices, relying parties may never receive updates.

In some contexts, other fields can provide a partial signal. For example, post-quantum-capable relying parties may be detected with the signature_algorithms and signature_algorithms_cert extensions. However, this relies on all post-quantum CAs being added at roughly the same time and that they are sufficiently interchangeable to be negotiated with these extensions. Trust anchor negotiation directly addresses this problem and allows for both gradual and possibly heterogeneous deployment of post-quantum CAs across relying parties.

8.2. Removing Untrustworthy CAs

When CAs are determined to be untrustworthy, relying parties must remove them to mitigate the risk to user security. Over time, this shrinks their intersection with older relying parties. Without negotiation, the result is authenticating parties have fewer and fewer CA choices available. Even determining the intersecting CAs can be difficult. Often, the only option is to try the new certificate and monitor errors. For authenticating parties that serve many diverse relying parties, this is a disruptive and risky process.

Trust anchor negotiation removes this constraint. If an authenticating party's CA is distrusted, it can use a new CA in addition to the existing one. The addition does not risk outages for older relying parties and may be chosen from a wider set of CAs, as it only needs to be compatible with the relying parties that distrusted the other CA.

Over time, the authenticating party can monitor which certificates it serves, and re-evaluate which CA or CAs to use. For example, it may find the new CA was sufficient, or that older relying parties have since all been updated. However, user security depends on the relying party's trust anchors, not the authenticating party's choice of CA, so this can occur asynchronously, based on serving needs and costs, rather than delay the response to a security incident.

8.3. Key Rotation

Despite the severity of root CA private key compromise and the benefits of routinely rotating cryptographic key material, such rotation in PKIs is often very rare. In 2023, the oldest root in [CHROME-ROOTS] and [MOZILLA-ROOTS] was 25 years old, dating to 1998.

Key rotation in PKIs used in TLS is challenging, as it combines the challenges described in both Section 8.1 and Section 8.2. Without trust anchor negotiation, authenticating parties cannot switch to the new root as long as any supported older relying party requires the old root. That, in turn, means relying parties cannot distrust the old root, leaving them vulnerable.

Trust anchor negotiation offers a smooth transition for CA key rotation. The CA can provide certification paths for the old and new roots. The authenticating party can then serve both paths without impacting older relying parties. New relying parties can then distrust the old root.

8.4. Other Root Transitions

The mechanisms in this document can aid PKI transitions beyond key rotation. For example, a CA operator may generate a postquantum root CA and issue from the classical and postquantum roots concurrently. The authenticating party will then, transparently and with no configuration change, serve both. As in Section 8.3, newer relying parties can then remove the classical roots, while older relying parties continue to function.

This same procedure may also be used to transition between newer, more size-efficient signature algorithms, as they are developed.

8.5. Intermediate Elision

In my PKIs, root CAs issue shorter-lived intermediate certificates which, in turn, issue end-entity certificates. This comes at a bandwidth cost: the TLS handshake includes an extra certificate, which includes a public key, signature, and X.509 metadata. Post-quantum signature algorithms will dramatically increase this cost. ML-DSA-65 [FIPS204], for example, has a total public key and signature size of 5,261 bytes.

[I-D.ietf-tls-cert-abridge] predistributes a specific set of intermediate CA certificates to relying parties so that these certificates can be omitted from TLS connections, as a compression scheme. Negotiating intermediate CAs as short-lived trust anchors also achieves this effect, but is usable by more relying parties than the specific intermediate set accommodates.

In this model, a CA operator provides authenticating parties with two certification paths: a longer path ending at a long-lived root and shorter path the other ending at a short-lived root. Relying parties trust both the long-lived root and the most recent short-lived root. The authenticating party sends the shorter path when possible, falling back to the longer path when the relying party's short-lived root is stale.

This achieves the same effect with a simpler and more flexible, general-purpose mechanism.

8.6. Conflicting Relying Party Requirements

An authenticating party may need to support relying parties with different, potentially conflicting requirements. For example, in contexts where online revocation checks are expensive, unreliable, or privacy-sensitive, user security is best served by short-lived certificates. In other contexts, long-lived certificates may be more appropriate for, e.g., systems that are offline for long periods of time or have unreliable clocks.

Trust anchor negotiation allows these conflicts to be resolved by different trust anchors where necessary. This avoids the need to compromise on user security or service availability.

8.7. Backup Certificates

An authenticating party may obtain certification paths from multiple CAs for redundancy. If one CA is compromised and removed from newer relying parties, the TLS server software will be able to gracefully serve a backup certification path, avoiding the immediate breakage that would otherwise be caused by this removal.

8.8. Public Key Pinning

To reduce security risk from misissued certificates, relying parties sometimes employ public key pinning [RFC7469]. Pinning effectively reduces a relying party's trust anchor list to a subset of the original set.

As other relying parties in the PKI evolve, the pinning relying party limits the authenticating party to satisfy both the pinning constraint and newer constraints in the PKI. This can lead to conflicts if, for example, the pinned CA is distrusted by a newer relying party. The authenticating party is then forced to either break the pinning relying party, or break the newer ones.

Trust anchor negotiation reduces this conflict, provided the pinning relying party negotiates with its reduced trust anchor list. The authenticating party can then use a certificate from the pinned CA with the pinning relying party, and another CA with other relying parties.

9. Privacy Considerations

9.1. Relying Parties

The trust_anchors extension is analogous to the certificate_authorities extension (Section 4.2.4 of [RFC8446]), but more size-efficient. Like certificate_authorities, trust_anchors reveals some information about the relying party's trust anchors. However, unlike certificate_authorities, trust_anchors allows a relying party to only reveal a trust anchor in response to the authenticating party's list, which reduces the fingerprinting exposure. This section provides guidance for a relying party to configure this mechanism, based on its privacy goals.

When using this extension, a relying party's trust anchors may be divided into three categories:

1. Trust anchors whose IDs the relying party never sends, but still trusts. These are trust anchors that do not participate in this mechanism.

2. Trust anchors whose IDs the relying party sends _conditionally_, i.e. only if the server offers them. For example, the relying party may indicate support for a trust anchor if its ID is listed in the server's HTTPS/SVCB record or trust anchor list in EncryptedExtensions.

3. Trust anchors whose IDs the relying party sends _unconditionally_, i.e. independently of the authenticating party's behavior.

Each of these categories carries a different fingerprinting exposure:

Trust anchors that do not participate are not revealed by this extension. However, they have some fingerprinting exposure due to being trusted. Given a certification path, an authenticating party can probe whether the relying party trusts the trust anchor by seeing if the relying party accepts it.

Trust anchor IDs sent in response to the authenticating party can only be observed actively. That is, the authenticating party could vary its list and observe how the client responds, in order to probe

for the client's trust anchor list. This is similar to the exposure of trust anchors not participating in this extension, except that the trust anchor can be probed by only knowing the trust anchor ID.

Trust anchor IDs sent unconditionally can be observed passively. This mode is analogous to the certificate_authorities extension. Relying parties SHOULD NOT unconditionally advertise trust anchor lists that are unique to an individual user. Rather, unconditionally-advertised lists SHOULD be empty or computed only from the trust anchors common to the relying party's anonymity set (Section 3.3 of [RFC6973]).

Relying parties SHOULD determine which trust anchors participate in this mechanism, and whether to advertise them unconditionally or conditionally, based on their privacy goals. PKIs that reliably use the DNS service parameter (Section 6) can rely on conditional advertisement for stronger privacy properties without a round-trip penalty.

Additionally, a relying party that computes the trust_anchors extension based on prior state may allow observers to correlate across connections. Relying parties SHOULD NOT maintain such state across connections that are intended to be uncorrelated. As above, implementing the DNS service parameter can avoid a round-trip penalty without such state.

9.2. Authenticating Parties

If the authenticating party is a server, the mechanisms in Section 6 and Section 4.3 enumerate the trust anchors for the server's available certification paths. This mechanism assumes they are not sensitive. Servers SHOULD NOT use this mechanism to negotiate certification paths with sensitive trust anchors.

In servers that host multiple services, this protocol only enumerates certification paths for the requested service. If, for example, a server uses the server_name extension to select services, the addition to EncryptedExtensions in Section 4.3 is expected to be filtered by server_name. Likewise, the DNS parameter in Section 6 only contains information for the corresponding service. In both cases, co-located services are not revealed.

The above does not apply if the authenticating party is a client. This protocol does not enumerate the available certification paths for a client.

10. Security Considerations

10.1. Incorrect Selection Metadata

If the authenticating party has provisioned certification paths with incorrect trust anchor IDs, it may negotiate inaccurately and send an untrusted path to the relying party when another candidate would have been trusted. This will not result in the untrusted path becoming trusted, but the connection will fail.

10.2. Trust Anchor Negotiation

Both the trust_anchors and certificate_authorities (Section 4.2.4 of [RFC8446]) extensions implement trust anchor negotiation, so security considerations are largely unchanged from certificate_authorities. This section discusses security considerations for trust anchor negotiation in general.

10.2.1. Relying Party Policies

PKI-based TLS authentication depends on the relying party's certificate policies. If the relying party trusts an untrustworthy CA, that CA can intercept TLS connections made by that relying party by issuing certificates associating the target name with the wrong TLS key.

This attack vector is available with or without trust anchor negotiation. The negotiation mechanism described in this document allows certificate selection to reflect a relying party's certificate policies. It does not determine the certificate policies themselves. Relying parties remain responsible for trusting only trustworthy CAs, and untrustworthy CAs remain a security risk when trusted.

10.2.2. Agility

As with other TLS parameters, negotiation reduces a conflict between availability and security, which allows PKIs to better mitigate security risks to users. When relying parties in an existing TLS ecosystem improve their certificate policies, trust anchor negotiation helps authenticating parties navigate differences between those relying parties and existing relying parties. Each set of requirements may be satisfied without compatibility risk to the other. Section 8 discusses such scenarios in more detail.

Negotiation also reduces pressures on relying parties to sacrifice user security for compatibility. If a relying party does not trust an authenticating party's current CA, connections between the two will fail until either the relying party trusts the CA or the authenticating party uses an already trusted CA. Without trust anchor negotiation, the authenticating party is limited to one

certificate, and therefore switching CAs risks compatibility problems with other relying parties. The relying party then faces compatibility pressure to add this CA, even if it deems the CA a security risk. With trust anchor negotiation, the authenticating party can use its existing CA _in addition to_ another CA trusted by the relying party. This allows the ecosystem to improve interoperability without sacrificing user security.

10.2.3. Serving Multiple Certificates

Trust anchor negotiation reduces compatibility pressures against authenticating parties serving certificates from a less common CA, as they can be served with other certificates. In some cases, the CA may have been distrusted, but still used to support older relying parties. As discussed in Section 8 and Section 10.2.2, this capability aids PKI transitions that mitigate security risks to users.

Even if the CA is untrustworthy, these certificates do not enable the CA to decrypt or intercept the connection. If a certificate asserts the correct information about the authenticating party, notably the correct public key, the authenticating party can safely present it. Issuing a certificate for the authenticating party's public key does not grant the CA access to the corresponding private key. Conversely, if the attacker already has access to the authenticating party's private key, they do not need to be in control of a CA to intercept a connection.

Rather, it is the relying party's choice of trusted CAs that determines susceptibility to interception. If the relying party trusts a misbehaving or attacker-controlled CA, the attacker can intercept the connection with a public key certified by that CA, regardless of which CA is used by the intended authenticating party. Conversely, if the relying party does not trust the attacker's CA, the attacker cannot successfully intercept the connection using a public key certified by this CA.

Choosing trusted CAs is a complex, security-critical process, the full considerations of which are outside the scope of this document. Relying parties thus SHOULD NOT interpret the authenticating party's choice of CA as an endorsement of the CA. Trusting a CA means trusting _all_ certificates issued by that CA, so it is not enough to observe correct certificates from an authenticating party. An untrustworthy CA may sign one correct certificate, but also sign incorrect certificates, possibly in the future, that can attack the relying party.

10.2.4. Targeting TLS Interception

A network attacker in possession of a misissued certificate could use trust anchor negotiation to differentiate clients and only enable TLS interception with clients that accept the certificate. The network attacker may wish to do this to reduce the odds of detection.

However, trust anchor negotiation only impacts detection where this differentiation was not already possible. In TLS, the client offers all its available TLS features, including cipher suites and other extensions, in the TLS ClientHello. Any variation in client TLS policies, related or unrelated to trust anchors, may be used as a fingerprint. Transport properties, such as IP geolocation, may also be used. While fingerprinting's heuristic nature makes broad, legitimate use difficult, a network attacker's single interception service can easily use it for targeted attacks.

If the attacker targets any clients that enforce Certificate Transparency [RFC6962], the misissued certificates will need to be publicly logged. In this case, detection is more robust, and client differentiation, with or without trust anchor negotiation, has no significant impact.

11. IANA Considerations

11.1. TLS ExtensionType Updates

IANA is requested to create the following entry in the TLS ExtensionType Values registry, defined by [RFC8446]:

Value	Extension Name	TLS 1.3	DTLS-Only	Recommended	Reference
TBD	trust_anchors	CH, EE, CR, CT	N	Y	[this-RFC]

Table 1

11.2. Media Type Updates

IANA is requested to create the following entry in the "Media Types" registry, defined in [RFC6838]:

Type name: application

Subtype name: pem-certificate-chain-with-properties

Required parameters: None

Optional parameters: None

Encoding considerations: 7bit

Security considerations: Carries a cryptographic certificate and its associated certificate chain and additional properties. This media type carries no active content.

Interoperability considerations: None

Published specification: [this-RFC, Section 7.3]

Applications that use this media type: ACME clients and servers, HTTP servers, other applications that need to be configured with a certificate chain

Additional information: Deprecated alias names for this type: n/a
Magic number(s): n/a
File extension(s): .pem
Macintosh file type code(s): n/a

Person & email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: n/a

Author: See Authors' Addresses section.

Change controller: IETF

11.3. PKIX Registry Updates

IANA is requested to create the following entry in the SMI Security for PKIX Module Identifier registry, defined by [RFC7299]:

Decimal	Description	References
TBD	id-mod-trustAnchorIDs-2025	[this-RFC]

Table 2

IANA is requested to create the following entry in the SMI Security for PKIX Certificate Extension registry, defined by [RFC7299]:

Decimal	Description	References
TBD	id-pe-trustAnchorID	[this-RFC]

Table 3

11.4. CertificatePropertyType Registry

[[TODO: Establish a CertificatePropertyType registry.]]

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4158] Cooper, M., Dzambasow, Y., Hesse, P., Joseph, S., and R. Nicholas, "Internet X.509 Public Key Infrastructure: Certification Path Building", RFC 4158, DOI 10.17487/RFC4158, September 2005, <https://www.rfc-editor.org/rfc/rfc4158>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC7299] Housley, R., "Object Identifier Registry for the PKIX Working Group", RFC 7299, DOI 10.17487/RFC7299, July 2014, <https://www.rfc-editor.org/rfc/rfc7299>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/rfc/rfc7468>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/rfc/rfc8555>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9371] Baber, A. and P. Hoffman, "Registration Procedures for Private Enterprise Numbers (PENs)", RFC 9371, DOI 10.17487/RFC9371, March 2023, <https://www.rfc-editor.org/rfc/rfc9371>.

[RFC9460] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/rfc/rfc9460>.

[X680] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ISO/IEC 8824-1:2021, 2021.

[X690] ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ISO/IEC 8825-1:2021, 2021.

12.2. Informative References

[CHROME-ROOTS]
Chromium, "Chrome Root Store", 30 August 2023, <https://chromium.googlesource.com/chromium/src/+/main/net/data/ssl/chrome_root_store>.

[FIPS204] National Institute of Standards and Technology (NIST), "Module-Lattice-based Digital Signature Standard", FIPS PUB 204, August 2023, <https://csrc.nist.gov/projects/post-quantum-cryptography>.

[I-D.ietf-plants-merkle-tree-certs]
Benjamin, D., O'Brien, D., Westerbaan, B., Valenta, L., and F. Valsorda, "Merkle Tree Certificates", Work in Progress, Internet-Draft, draft-ietf-plants-merkle-tree-certs-03, 20 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-plants-merkle-tree-certs-03>.

[I-D.ietf-tls-cert-abridge]
Jackson, D., "Abridged Compression for WebPKI Certificates", Work in Progress, Internet-Draft, draft-ietf-tls-cert-abridge-02, 16 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-cert-abridge-02>.

[MOZILLA-ROOTS]
Mozilla, "Mozilla Included CA Certificate List", 30 August 2023, <https://wiki.mozilla.org/CA/Included_Certificates>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/rfc/rfc6962>.

[RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <https://www.rfc-editor.org/rfc/rfc7469>.

[SCTNotAfter]
Adrian, D., "How to distrust a CA without any certificate errors", March 2025, <https://dadrian.io/blog/posts/sct-not-after/>.

Appendix A. ASN.1 Module

```
TrustAnchorIDs-2025
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-trustAnchorIDs-2025(TBD) }

DEFINITIONS EXPLICIT TAGS ::=
BEGIN

IMPORTS
    EXTENSION
    FROM PKIX-CommonTypes-2009 -- From [RFC5912]
    { iso(1) identified-organization(3) dod(6)
      internet(1) security(5) mechanisms(5) pkix(7)
      id-mod(0) id-mod-pkixCommon-02(57) };

-- Trust Anchor IDs Certificate Extension

ext-TrustAnchorID EXTENSION ::= {
    SYNTAX TrustAnchorID
    IDENTIFIED BY id-pe-trustAnchorID
    CRITICALITY { FALSE } }

id-pe-trustAnchorID OBJECT IDENTIFIER ::=
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-pe(1) TBD }

TrustAnchorID ::= RELATIVE-OID

END
```

Acknowledgements

The authors thank Nick Harper, and Emily Stark for many valuable discussions and insights which led to this document. Thanks also to Aaron Gable for providing feedback on ACME extensions.

Authors' Addresses

Bob Beck
OpenSSL
Email: beck@openssl.org

David Benjamin
Google LLC
Email: davidben@google.com

Devon O’Brien
Email: devon.obrien@gmail.com

Kyle Nekritz
Meta
Email: knekritz@meta.com