

Transport Layer Security  
Internet-Draft  
Intended status: Standards Track  
Expires: 15 November 2025

B. Beck  
D. Benjamin  
D. O'Brien  
Google LLC  
K. Nekritz  
Meta  
14 May 2025

TLS Trust Anchor Identifiers  
draft-ietf-tls-trust-anchor-ids-01

## 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.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 15 November 2025.

## Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Conventions and Definitions . . . . .	5
2.1. Terminology and Roles . . . . .	5
3. Trust Anchor Identifiers . . . . .	5
3.1. Relying Party Configuration . . . . .	6
3.2. Authenticating Party Configuration . . . . .	7
4. TLS Extension . . . . .	7
4.1. Overview . . . . .	7
4.2. Certificate Selection . . . . .	8
4.3. Retry Mechanism . . . . .	9
5. DNS Service Parameter . . . . .	11
5.1. Syntax . . . . .	11
5.2. Configuring Services . . . . .	12
5.3. Client Behavior . . . . .	12
6. Certificate Properties . . . . .	12
6.1. Media Type . . . . .	13
6.2. ACME Extension . . . . .	14
7. Use Cases . . . . .	14
7.1. Making Use of Newly-Trusted CAs . . . . .	15
7.2. Removing Untrustworthy CAs . . . . .	16

7.3.	Key Rotation . . . . .	16
7.4.	Other Root Transitions . . . . .	17
7.5.	Intermediate Elision . . . . .	17
7.6.	Conflicting Relying Party Requirements . . . . .	18
7.7.	Backup Certificates . . . . .	18
7.8.	Public Key Pinning . . . . .	18
8.	Privacy Considerations . . . . .	18
8.1.	Relying Parties . . . . .	19
8.2.	Authenticating Parties . . . . .	20
9.	Security Considerations . . . . .	20
9.1.	Incorrect Selection Metadata . . . . .	20
9.2.	Trust Anchor Negotiation . . . . .	21
9.2.1.	Relying Party Policies . . . . .	21
9.2.2.	Agility . . . . .	21
9.2.3.	Serving Multiple Certificates . . . . .	22
9.2.4.	Targeting TLS Interception . . . . .	22
10.	IANA Considerations . . . . .	23
10.1.	TLS ExtensionType Updates . . . . .	23
10.2.	Media Type Updates . . . . .	23
10.3.	CertificatePropertyType Registry . . . . .	24
11.	References . . . . .	24
11.1.	Normative References . . . . .	24
11.2.	Informative References . . . . .	26
Appendix A.	ASN.1 Module . . . . .	26
Acknowledgements	. . . . .	27
Authors' Addresses	. . . . .	27

## 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 certificate 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 three parts to this mechanism:

1. Section 3 defines \_trust anchor IDs\_, which are short, unique identifiers for X.509 trust anchors.
2. Section 4 defines a TLS extension that communicates the relying party's requested trust anchors, and the authenticating party's available ones. When the relying party is a TLS client, it can mitigate large lists by sending a, possibly empty, subset of its trust anchors to the TLS server. The server provides its list of available trust anchors in response so that the client can retry on mismatch.
3. Section 5 allows TLS servers to advertise their available trust anchors in HTTPS or SVCB [RFC9460] DNS records. TLS clients can then 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 for a trust anchor. To simplify allocation, trust anchor IDs are defined with object identifiers (OIDs) [X680] and IANA-registered Private Enterprise Numbers (PENs) [RFC9371]:

A trust anchor ID is defined with an OID under the OID arc of some PEN. 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.

Trust anchor IDs SHOULD be allocated by the CA operator and 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.

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

### 3.1. 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-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-trustAnchorID OBJECT IDENTIFIER ::= { 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 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.

### 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 which participates in this protocol must be configured with the trust anchor ID for its corresponding trust anchor. It is RECOMMENDED, though not required, that this information come from the CA. Section 6 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 associated trust anchor IDs, but such paths will not participate in this protocol. Those paths 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>;
```

```
TrustAnchorID TrustAnchorIDList<0..2^16-1>;
```

When the extension is sent in the ClientHello or CertificateRequest messages, the extension\_data is a TrustAnchorIDList and indicates that the sender supports the specified trust anchors. 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 a TrustAnchorIDList containing the list of trust anchors that server has available, in the server's 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 whose trust anchor ID appears 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 8), 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 5 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 5 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. 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.

### 5.1. 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,
```

## 5.2. 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.

## 5.3. 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.

## 6. 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), (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 a single property, `trust_anchor_id`. The data field of the property contains the binary representation of the trust anchor ID of the certification path's trust anchor, as described in Section 3.2. Future documents may define other properties for use with other mechanisms.

Authenticating parties MUST ignore properties with unrecognized `CertificatePropertyType` values.

### 6.1. 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.

`CertificatePropertyList`s are encoded using the "CERTIFICATE PROPERTIES" label. The encoded data is a serialized `CertificatePropertyList`, defined in Section 6.

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 10.2.

## 6.2. ACME Extension

The format defined in Section 6.1 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 a trust\_anchor\_id property 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.

## 7. 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.

#### 7.1. 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.

## 7.2. 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.

## 7.3. 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 7.1 and Section 7.2. 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.



#### 7.4. 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 7.3, 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.

#### 7.5. 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.

### 7.6. 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.

### 7.7. Backup Certificates

An authenticating party may obtain certificate 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 certificate path, avoiding the immediate breakage that would otherwise be caused by this removal.

### 7.8. 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.

## 8. Privacy Considerations

### 8.1. 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 5) 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.

## 8.2. Authenticating Parties

If the authenticating party is a server, the mechanisms in Section 5 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 5 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.

## 9. Security Considerations

### 9.1. 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.

## 9.2. 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.

### 9.2.1. 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.

### 9.2.2. 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 7 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.

### 9.2.3. 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 7 and Section 9.2.2, 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.

### 9.2.4. 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.

## 10. IANA Considerations

### 10.1. 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

### 10.2. 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 6.1]

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

### 10.3. CertificatePropertyType Registry

[[TODO: Establish a CertificatePropertyType registry.]]

## 11. References

### 11.1. 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>>.
- [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.

## 11.2. Informative References

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

## Appendix A. ASN.1 Module

```
TrustAnchorIDs DEFINITIONS ::= BEGIN

id-trustAnchorID OBJECT IDENTIFIER ::= { 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  
Google LLC  
Email: bbe@google.com

David Benjamin  
Google LLC  
Email: davidben@google.com

Devon O'Brien  
Google LLC  
Email: asymmetric@google.com

Kyle Nekritz  
Meta  
Email: knekritz@meta.com