Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 31, 2012

P. Hoffman
VPN Consortium
J. Schlyter
Kirei AB

April 29, 2012

The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA
draft-ietf-dane-protocol-20

Abstract

Encrypted communication on the Internet often uses Transport Level Security (TLS), which depends on third parties to certify the keys used. This document improves on that situation by enabling the administrators of domain names to specify the keys used in that domain's TLS servers. This requires matching improvements in TLS client software, but no change in TLS server software.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. Background and Motivation

Applications that communicate over the Internet often need to prevent eavesdropping, tampering, or forgery of their communications. The Transport Layer Security (TLS) protocol provides this kind of communications security over the Internet, using channel encryption.

The security properties of encryption systems depend strongly on the keys that they use. If secret keys are revealed, or if public keys can be replaced by bogus keys, these systems provide little or no security.

TLS uses certificates to bind keys and names. A certificate combines a published key with other information such as the name of the service that uses the key, and this combination is digitally signed by another key. Having a certificate for a key is only helpful if one trusts the other key that signed the certificate. If that other key was itself revealed or substituted, then its signature is worthless in proving anything about the first key.

On the Internet, this problem has been solved for years by entities called "Certification Authorities" (CAs). CAs protect their secret key vigorously, while supplying their public key to the software vendors who build TLS clients. They then sign certificates, and supply those to TLS servers. TLS client software uses a set of these CA keys as "trust anchors" to validate the signatures on certificates that the client receives from TLS servers. Client software typically allows any CA to usefully sign any other certificate.

The public CA model upon which TLS has depended is fundamentally vulnerable because it allows any of these CAs to issue a certificate for any domain name. A single trusted CA that betrays its trust, either voluntarily or by providing less-than-vigorous protection for its secrets and capabilities, can undermine the security offered by any certificates employed with TLS. This problem arises because a compromised CA can issue a replacement certificate that contains a bogus key (that is, a key not corresponding to the entity identified in the certificate). Recent experiences with compromises of CAs or their trusted partners have led to very serious security problems, such as the governments of multiple countries attempting to wiretap and/or subvert major TLS-protected web sites trusted by millions of users.

The DNS Security Extensions (DNSSEC) provides a similar model that involves trusted keys signing the information for untrusted keys. However, DNSSEC provides three significant improvements. Keys are

tied to names in the Domain Name System (DNS), rather than to arbitrary identifying strings; this is more convenient for Internet protocols. Signed keys for any domain are accessible online through a straightforward query using the standard DNSSEC protocol, so there is no problem distributing the signed keys. Most significantly, the keys associated with a domain name can only be signed by a key associated with the parent of that domain name; for example, the keys for "example.com" can only be signed by the keys for "com", and the keys for "com" can only be signed by the DNS root. This prevents an untrustworthy signer from compromising anyone's keys except those in their own subdomains. Like TLS, DNSSEC relies on public keys that come built into the DNSSEC client software, but these keys come only from a single root domain rather than from a multiplicity of CAs.

DNS-Based Authentication of Named Entities (DANE) offers the option to use the DNSSEC infrastructure to store and sign keys and certificates that are used by TLS. DANE is envisioned as a preferable basis for binding public keys to DNS names, because the entities that vouch for the binding of public key data to DNS names are the same entities responsible for managing the DNS names in question. While the resulting system still has residual security vulnerabilities, it restricts the scope of assertions that can be made by any entity, consistent with the naming scope imposed by the DNS hierarchy. As a result, DANE embodies the security "principle of least privilege" that is lacking in the current public CA model.

1.2. Securing the Association of a Domain Name with a Server's Certificate

A TLS client begins a connection by exchanging messages with a TLS server. For many application protocols, it looks up the server's name using the DNS to get an Internet Protocol (IP) address associated with the name. It then begins a connection to a client-define port at that address, and sends an initial message there. However, the client does not yet know whether an adversary is intercepting and/or altering its communication before it reaches the TLS server. It does not even know whether the real TLS server associated with that domain name has ever received its initial messages.

The first response from the server in TLS may contain a certificate. In order for the TLS client to authenticate that it is talking to the expected TLS server, the client must validate that this certificate is associated with the domain name used by the client to get to the server. Currently, the client must extract the domain name from the certificate and must successfully validate the certificate, including chaining to a trust anchor.

There is a different way to authenticate the association of the server's certificate with the intended domain name without trusting an external CA. Given that the DNS administrator for a domain name is authorized to give identifying information about the zone, it makes sense to allow that administrator to also make an authoritative binding between the domain name and a certificate that might be used by a host at that domain name. The easiest way to do this is to use the DNS, securing the binding with DNSSEC.

There are many use cases for such functionality. [RFC6394] lists the ones to which the DNS RRtype in this document apply. [RFC6394] also lists many requirements, most of which this document is believed to meet. Section 5 covers the applicability of this document to the use cases in detail. The protocol in this document can generally be referred to as the "DANE TLSA" protocol. ("TLSA" does not stand for anything; it is just the name of the RRtype.)

This document applies to both TLS [RFC5246] and DTLS [RFC6347]. In order to make the document more readable, it mostly only talks about "TLS", but in all cases, it means "TLS or DTLS". This document only relates to securely associating certificates for TLS and DTLS with host names; other security protocols are handled in other documents. For example, keys for IPsec are covered in [RFC4025] and keys for SSH are covered in [RFC4255]. Although the references in this paragraph are to TLS and DTLS version 1.2, the DANE TLSA protocol can also be used with earlier versions of TLS and DTLS.

1.3. Method For Securing Certificate Associations

A certificate association is formed from a piece of information identifying a certificate and the domain name where the server application runs. The combination of a trust anchor and a domain name can also be a certificate association.

A DNS query can return multiple certificate associations, such as in the case of a server that is changing from one certificate to another (described in more detail in Appendix A.4).

This document only applies to PKIX [RFC5280] certificates, not certificates of other formats. Later updates to this document might specify how to use certificates in other formats.

This document defines a secure method to associate the certificate that is obtained from the TLS server with a domain name using DNS; the DNS information needs to be be protected by DNSSEC. Because the certificate association was retrieved based on a DNS query, the domain name in the query is by definition associated with the certificate. Note that this document does not cover how to associate

certificates with domain names for application protocols that depend on SRV, NAPTR, and similar DNS resource records; it is expected that later updates to this document might cover methods for making those associations.

DNSSEC, which is defined in [RFC4033], [RFC4034], and [RFC4035], uses cryptographic keys and digital signatures to provide authentication of DNS data. Information that is retrieved from the DNS and that is validated using DNSSEC is thereby proved to be the authoritative data. The DNSSEC signature needs to be validated on all responses that use DNSSEC in order to assure the proof of origin of the data.

This document does not specify how DNSSEC validation occurs because there are many different proposals for how a client might get validated DNSSEC results, such as from a DNSSEC-aware resolver that is coded in the application, from a trusted DNSSEC resolver on the machine on which the application is running, or from a trusted DNSSEC resolver with which the application is communicating over an authenticated and integrity-protected channel or network. This is described in more detail in Section 7 of [RFC4033].

This document only relates to getting the DNS information for the certificate association securely using DNSSEC; other secure DNS mechanisms are out of scope.

1.4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document also makes use of standard PKIX, DNSSEC, TLS, and DNS terminology. See [RFC5280], [RFC4033], [RFC5246], and [STD13] respectively, for these terms. In addition, terms related to TLS-protected application services and DNS names are taken from [RFC6125].

2. The TLSA Resource Record

The TLSA DNS resource record (RR) is used to associate a TLS server certificate or public key with the domain name where the record is found, thus forming a "TLS certificate association". The semantics of how the TLSA RR is interpreted are given later in this document.

The type value for the TLSA RR type is defined in Section 7.1.

The TLSA RR is class independent.

The TLSA RR has no special TTL requirements.

2.1. TLSA RDATA Wire Format

The RDATA for a TLSA RR consists of a one octet certificate usage field, a one octet selector field, a one octet matching type field and the certificate association data field.

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Cert. Usage  |   Selector    | Matching Type |               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               /
/                                                               /
/                 Certificate Association Data                  /
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

2.1.1. The Certificate Usage Field

A one-octet value, called "certificate usage", specifies the provided association that will be used to match the certificate presented in the TLS handshake. This value is defined in a new IANA registry (see Section 7.2) in order to make it easier to add additional certificate usages in the future. The certificate usages defined in this document are:

0 -- Certificate usage 0 is used to specify a CA certificate, or the public key of such a certificate, that MUST be found in any of the PKIX certification paths for the end entity certificate given by the server in TLS. This certifcate usage is sometimes referred to as "CA constraint" because it limits which CA can be used to issue certificates for a given service on a host. The presented certificate MUST pass PKIX certification path validation and a CA certificate that matches the TLSA record MUST be included as part of a valid certification path. Because this certifcate usage allows both trust anchors and CA certificates, the certificate might or might not have the basicConstraints extension present.

1 -- Certificate usage 1 is used to specify an end entity certificate, or the public key of such a certificate, that must be matched with the end entity certificate given by the server in TLS. This certifcate usage is sometimes referred to as "service certificate constraint" because it limits which end entity certificate can be used by a given service on a host. The target certificate MUST pass PKIX certification path validation and MUST match the TLSA record.

2 -- Certificate usage 2 is used to specify a certificate, or the public key of such a certificate, that must be used as the trust anchor when validating the end entity certificate given by the server in TLS. This certifcate usage is sometimes referred to as "trust anchor assertion" and allows a domain name administrator to specify a new trust anchor. For example, if the domain issues its own certificates under its own CA that is not expected to be in the end users' collection of trust anchors. The target certificate MUST pass PKIX certification path validation, with any certificate matching the TLSA record considered to be a trust anchor for this certification path validation.

3 -- Certificate usage 3 is used to specify a certificate, or the public key of such a certificate, that must match the end entity certificate given by the server in TLS. This certifcate usage is sometimes referred to as "domain-issued certificate" because it allows for a domain name administrator to issue certificates for a domain without involving a third-party CA. The target certificate MUST match the TLSA record. The difference between certificate usage 1 and certificate usage 3 is that certificate usage 1 requires that the certificate pass PKIX validation, but PKIX validation is not tested for certificate usage 3.

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

2.1.2. The Selector Field

A one-octet value, called "selector", specifies which part of the TLS certificate presented by the server will be matched against the association data. This value is defined in a new IANA registry (see Section 7.3. The selectors defined in this document are:

0 -- Full certificate

1 -- SubjectPublicKeyInfo

The SubjectPublicKeyInfo is a binary structure defined in [RFC5280].

(Note that the use of "selector" in this document is completely unrelated to the use of "selector" in DKIM [RFC6376].)

2.1.3. The Matching Type Field

A one-octet value, called "matching type", specifies how the certificate association is presented. This value is defined in a new IANA registry (see Section 7.4). The types defined in this document are:

0 -- Exact match on selected content

1 -- SHA-256 hash of selected content [RFC6234]

2 -- SHA-512 hash of selected content [RFC6234]

If the TLSA record's matching type is a hash, having the record use the same hash algorithm that was used in the signature in the certificate (if possible) will assist clients that support a small number of hash algorithms.

2.1.4. The Certificate Association Data Field

The "certificate association data" to be matched. These bytes are either raw data (that is, the full certificate or its SubjectPublicKeyInfo, depending on the selector) for matching type 0, or the hash of the raw data for matching types 1 and 2. The data refers to the certificate in the association, not to the TLS ASN.1 Certificate object.

2.2. TLSA RR Presentation Format

The presentation format of the RDATA portion (as defined in [RFC1035]) is as follows:

- The certificate usage field MUST be represented as an 8-bit unsigned decimal integer.

- The selector field MUST be represented as an 8-bit unsigned decimal integer.

- The matching type field MUST be represented as an 8-bit unsigned decimal integer.

- The certificate association data field MUST be represented as a string of hexadecimal characters. Whitespace is allowed within the string of hexadecimal characters.

2.3. TLSA RR Examples

In the following examples, the domain name is formed using the rules in Section 3.

An example of a hashed (SHA-256) association of a PKIX CA certificate:

```
_443._tcp.www.example.com. IN TLSA (
   0 0 1 d2abde240d7cd3ee6b4b28c54df034b9
         7983a1d16e8a410e4561cb106618e971 )
```

An example of a hashed (SHA-512) subject public key association of a PKIX end entity certificate:

```
_443._tcp.www.example.com. IN TLSA (
   1 1 2 92003ba34942dc74152e2f2c408d29ec
         a5a520e7f2e06bb944f4dca346baf63c
         1b177615d466f6c4b71c216a50292bd5
         8c9ebdd2f74e38fe51ffd48c43326cbc )
```

An example of a full certificate association of a PKIX end entity certificate:

```
_443._tcp.www.example.com. IN TLSA (
   3 0 0 30820307308201efa003020102020... )
```

3. Domain Names for TLSA Certificate Associations

Unless there is a protocol-specific specification that is different than this one, TLSA resource records are stored at a prefixed DNS domain name. The prefix is prepared in the following manner:

1. The decimal representation of the port number on which a TLS-based service is assumed to exist is prepended with an underscore character ("_") to become the left-most label in the prepared domain name. This number has no leading zeros.

2. The protocol name of the transport on which a TLS-based service is assumed to exist is prepended with an underscore character ("_") to become the second left-most label in the prepared domain name. The transport names defined for this protocol are "tcp", "udp" and "sctp".

3. The domain name is appended to the result of step 2 to complete the prepared domain name.

For example, to request a TLSA resource record for an HTTP server running TLS on port 443 at "www.example.com", "_443._tcp.www.example.com" is used in the request. To request a TLSA resource record for an SMTP server running the STARTTLS protocol on port 25 at "mail.example.com", "_25._tcp.mail.example.com" is used.

4. Use of TLSA Records in TLS

Section 2.1 of this document defines the mandatory matching rules for the data from the TLSA certificate associations and the certificates received from the TLS server.

The TLS session that is to be set up MUST be for the specific port number and transport name that was given in the TLSA query.

Some specifications for applications that run over TLS, such as [RFC2818] for HTTP, require the server's certificate to have a domain name that matches the host name expected by the client. Some specifications such as [RFC6125] detail how to match the identity given in a PKIX certificate with those expected by the user.

If a TLSA record has certificate usage 2, the corresponding TLS server SHOULD send the certificate that is referenced just like it currently sends intermediate certificates.

4.1. Usable Certificate Associations

An implementation of this protocol makes a DNS query for TLSA records, validates these records using DNSSEC, and uses the resulting TLSA records and validation status to modify its responses to the TLS server.

Determining whether a TLSA RRset can be used MUST be based on the DNSSEC validation state (as defined in [RFC4033]).

- A TLSA RRset whose DNSSEC validation state is secure MUST be used as a certificate association for TLS unless a local policy would prohibit the use of the specific certificate association in the secure TLSA RRset.

- If the DNSSEC validation state on the response to the request for the TLSA RRset is bogus, this MUST cause TLS not to be started or, if the TLS negotiation is already in progress, MUST cause the connection to be aborted.

- A TLSA RRset whose DNSSEC validation state is indeterminate or insecure cannot be used for TLS and MUST be considered unusable.

Clients which validate the DNSSEC signatures themselves MUST use standard DNSSEC validation procedures. Clients that rely on another entity to perform the DNSSEC signature validation MUST use a secure mechanism between themselves and the validator. Examples of secure transports to other hosts include TSIG [RFC2845], SIG(0) [RFC2931], and IPsec [RFC6071]. Note that it is not sufficient to use secure transport to a DNS resolver that does not do DNSSEC signature validation.

If a certificate association contains a certificate usage, selector, or matching type that is not understood by the TLS client, that certificate association MUST be considered unusable. If the comparison data for a certificate is malformed, the certificate association MUST be considered unusable.

If a certificate association contains a matching type or certificate association data that uses a cryptographic algorithm that is considered too weak for the TLS client's policy, the certificate association MUST be considered unusable.

If an application receives zero usable certificate associations from a DNS request or from its cache, it processes TLS in the normal fashion without any input from the TLSA records. If an application receives one or more usable certificate associations, it attempts to match each certificate association with the TLS server's end entity certificate until a successful match is found. During the TLS handshake, if none of the certificate associations matches the certificate given by the TLS server, the TLS client MUST abort the handshake.

The application can perform the TLSA lookup before initiating the TLS handshake, or do it during the TLS handshake: the choice is up to the client.

5. TLSA and DANE Use Cases and Requirements

The different types of certificate associations defined in TLSA are matched with various sections of [RFC6394]. The use cases from Section 3 of [RFC6394] are covered in this document as follows:

3.1 CA Constraints -- Implemented using certificate usage 0.

3.2 Certificate Constraints -- Implemented using certificate usage 1.

3.3 Trust Anchor Assertion and Domain-Issued Certificates -- Implemented using certificate usages 2 and 3, respectively.

The requirements from Section 4 of [RFC6394] are covered in this document as follows:

Multiple Ports -- The TLSA records for different application services running on a single host can be distinguished through the service name and port number prefixed to the host name (see Section 3).

No Downgrade -- Section 4 specifies the conditions under which a client can process and act upon TLSA records. Specifically, if the DNSSEC status for the TLSA resource record set is determined to be bogus, the TLS connection (if started) will fail.

Encapsulation -- Covered in the TLSA response semantics.

Predictability -- The appendices of this specification provide operational considerations and implementation guidance in order to enable application developers to form a consistent interpretation of the recommended client behavior.

Opportunistic Security -- If a client conformant to this specification can reliably determine the presence of a TLSA record, it will attempt to use this information. Conversely, if a client can reliably determine the absence of any TLSA record, it will fall back to processing TLS in the normal fashion. This is discussed in Section 4.

Combination -- Multiple TLSA records can be published for a given host name, thus enabling the client to construct multiple TLSA certificate associations that reflect different assertions. No support is provided to combine two TLSA certificate associations in a single operation.

Roll-over -- TLSA records are processed in the normal manner within the scope of DNS protocol, including the TTL expiration of the records. This ensures that clients will not latch onto assertions made by expired TLSA records, and will be able to transition from using one public key or certificate usage to another.

Simple Key Management -- The SubjectPublicKeyInfo selector in the TLSA record provides a mode that enables a domain holder to only have to maintain a single long-lived public/private key pair without the need to manage certificates. Appendix A outlines the usefulness and the potential downsides to using this mode.

Minimal Dependencies -- This specification relies on DNSSEC to protect the origin authenticity and integrity of the TLSA resource record set. Additionally, if DNSSEC validation is not performed on the system that wishes to use TLSA certificate bindings, this specification requires that the "last mile" be over a secure transport. There are no other deployment dependencies for this approach.

Minimal Options -- The operating modes map precisely to the DANE use cases and requirements. DNSSEC use is mandatory in that this specification encourages applications to use only those TLSA records that are shown to be validated.

Wild Cards -- Covered in a limited manner in the TLSA request syntax; see Appendix A.

Redirection -- Covered in the TLSA request syntax; see Appendix A.

6. Mandatory-to-Implement Features

TLS clients conforming to this specification MUST be able to correctly interpret TLSA records with certificate usages 0, 1, 2, and 3. TLS clients conforming to this specification MUST be able to compare a certificate association with a certificate from the TLS handshake using selector types 0 and 1, and matching type 0 (no hash used) and matching type 1 (SHA-256), and SHOULD be able to make such comparisons with matching type 2 (SHA-512).

At the time this is written, it is expected that there will be a new family of hash algorithms called SHA-3 within the next few years. It is expected that some of the SHA-3 algorithms will be mandatory and/or recommended for TLSA records after the algorithms are fully defined. At that time, this specification will be updated.

7. IANA Considerations

IANA is requested to make the assignments in this section. IANA might also consider making a "DANE" section in the main IANA registry to help developers find related registries that might be created in the future.

In the following sections, "RFC Required" was chosen for TLSA certficate usages and "Specification Required" for selectors and matching types because of the amount of detail that is likely to be needed for implementers to correctly implement new certificate usages as compared to new selectors and matching types.

7.1. TLSA RRtype

This document uses a new DNS RR type, TLSA, whose value (52) was allocated by IANA from the Resource Record (RR) TYPEs subregistry of the Domain Name System (DNS) Parameters registry.

7.2. TLSA Certificate Usages

This document creates a new registry, "Certificate Usages for TLSA Resource Records". The registry policy is "RFC Required". The initial entries in the registry are:

Value	Short description	Reference
0	CA constraint	[This]
1	Service certificate constraint	[This]
2	Trust anchor assertion	[This]
3	Domain-issued certificate	[This]
4-254	Unassigned	
255	Private use	

Applications to the registry can request specific values that have yet to be assigned.

7.3. TLSA Selectors

This document creates a new registry, "Selectors for TLSA Resource Records". The registry policy is "Specification Required". The initial entries in the registry are:

Value	Short description	Reference
0	Full Certificate	[This]
1	SubjectPublicKeyInfo	[This]
2-254	Unassigned	
255	Private use	

Applications to the registry can request specific values that have yet to be assigned.

7.4. TLSA Matching Types

This document creates a new registry, "Matching Types for TLSA Resource Records". The registry policy is "Specification Required". The initial entries in the registry are:

Value	Short description	Reference
0	No hash used	[This]
1	SHA-256	RFC 6234
2	SHA-512	RFC 6234
3-254	Unassigned	
255	Private use	

Applications to the registry can request specific values that have yet to be assigned.

8. Security Considerations

The security of the DNS RRtype described in this document relies on the security of DNSSEC to verify that the TLSA record has not been altered.

A DNS administrator who goes rogue and changes both the A, AAAA, and/or TLSA records for a domain name can cause the user to go to an unauthorized server that will appear authorized, unless the client performs PKIX certification path validation and rejects the certificate. That administrator could probably get a certificate issued by some CA anyway, so this is not an additional threat.

If the authentication mechanism for adding or changing TLSA data in a zone is weaker than the authentication mechanism for changing the A and/or AAAA records, a man-in-the-middle who can redirect traffic to their site may be able to impersonate the attacked host in TLS if they can use the weaker authentication mechanism. A better design for authenticating DNS would be to have the same level of authentication used for all DNS additions and changes for a particular domain name.

SSL proxies can sometimes act as a man-in-the-middle for TLS clients. In these scenarios, the clients add a new trust anchor whose private key is kept on the SSL proxy; the proxy intercepts TLS requests, creates a new TLS session with the intended host, and sets up a TLS session with the client using a certificate that chains to the trust anchor installed in the client by the proxy. In such environments, using TLSA records will prevent the SSL proxy from functioning as expected because the TLS client will get a certificate association

from the DNS that will not match the certificate that the SSL proxy uses with the client. The client, seeing the proxy's new certificate for the supposed destination will not set up a TLS session.

Client treatment of any information included in the certificate trust anchor is a matter of local policy. This specification does not mandate that such information be inspected or validated by the server's domain name administrator.

If a server's certificate is revoked, or if an intermediate CA in a chain between the end entity and a trust anchor has its certificate revoked, a TLSA record with a certificate type of 2 that matches the revoked certificate would in essence override the revocation because the client would treat that revoked certificate as a trust anchor and thus not check its revocation status. Because of this, domain administrators need to be responsible for being sure that the key or certificate used in TLSA records with a certificate type of 2 are in fact able to be used as reliable trust anchors.

Certificates that are delivered in TLSA with certificate usage 2 fundamentally change the way the TLS server's end entity certificate is evaluated. For example, the server's certificate might chain to an existing CA through an intermediate CA that has certain policy restrictions, and the certificate would not pass those restrictions and thus normally be rejected. That intermediate CA could issue itself a new certificate without the policy restrictions and tell its customers to use that certificate with certificate usage 2. This in essence allows an intermediate CA to be come a trust anchor for certificates that the end user might have expected to chain to an existing trust anchor.

If an administrator wishes to stop using a TLSA record, the administrator can simply remove it from the DNS. Normal clients will stop using the TLSA record after the TTL has expired. Replay attacks against the TLSA record are not possible after the expiration date on the RRsig of the TLSA record that was removed.

The client's full trust of a certificate retrieved from a TLSA record with a certificate usage of 2 or 3 may be a matter of local policy. While such trust is limited to the specific domain name for which the TLSA query was made, local policy may deny the trust or further restrict the conditions under which that trust is permitted.

8.1. Comparing DANE to Public CAs

Comparing the risk for current common TLS clients against the risk for DANE clients using external DNSSEC validators is difficult. The common model for TLS clients is that they trust a large number of

commercial CAs who can issue certificates for any domain name. A DANE-aware TLS client that that is trusting an external DNSSEC validator is as vulnerable as a DANE-unaware TLS client because any CA or any external validator can assert any key for any domain.

If it is less likely that a user will hear about its trusted DNSSEC validators being compromised that it is of a public CA being compromised, a client with an external validator could be worse off than a current TLS client that is depending on the current public CAs. A counter-argument to this is a single external DNSSEC validator is a much less interesting target than a public CA, particularly if many clients use their own DNSSEC validators or validators that reside on the computer on which they are running.

Current public CAs are assumed to have better defenses than current DNSSEC validators, but that perception cannot be proven one way or another. Similarly, if DNSSEC validation becomes more common after the release of this document, it cannot be predicted whether or not that will increase the level of security of DNSSEC validators more or less than the security of public CAs. Thus, it is difficult to foresee which system has a higher risk.

8.2. DNS Caching

Implementations of this protocol rely heavily on the DNS, and are thus prone to security attacks based on the deliberate mis-association of TLSA records and DNS names. Implementations need to be cautious in assuming the continuing validity of an assocation between a TLSA record and a DNS name.

In particular, implementations SHOULD rely on their DNS resolver for confirmation of an association between a TLSA record and a DNS name, rather than caching the result of previous domain name lookups. Many platforms already can cache domain name lookups locally when appropriate, and they SHOULD be configured to do so. It is proper for these lookups to be cached, however, only when the TTL (Time To Live) information reported by the DNS makes it likely that the cached information will remain useful.

If implementations cache the results of domain name lookups in order to achieve a performance improvement, they MUST observe the TTL information reported by DNS. Implementations that fail to follow this rule could be spoofed or have access denied when a previously-accessed server's TLSA record changes, such as during a certificate rollover.

9. Acknowledgements

Many of the ideas in this document have been discussed over many years. More recently, the ideas have been discussed by the authors and others in a more focused fashion. In particular, some of the ideas and words here originated with Paul Vixie, Dan Kaminsky, Jeff Hodges, Phill Hallam-Baker, Simon Josefsson, Warren Kumari, Adam Langley, Ben Laurie, Ilari Liusvaara, Ondrej Mikle, Scott Schmit, Ondrej Sury, Richard Barnes, Jim Schaad, Stephen Farrell, Suresh Krishnaswamy, Peter Palfrader, Pieter Lexis, Wouter Wijngaards, John Gilmore, and Murray Kucherawy.

This document has also been greatly helped by many active participants of the DANE Working Group.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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, May 2008.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[STD13] Mockapetris, P., "Domain Name System", STD 13, November 1987.

10.2. Informative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[RFC2931] Eastlake, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, September 2000.

[RFC4025] Richardson, M., "A Method for Storing IPsec Keying Material in DNS", RFC 4025, March 2005.

[RFC4255] Schlyter, J. and W. Griffin, "Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints", RFC 4255, January 2006.

[RFC4641] Kolkman, O. and R. Gieben, "DNSSEC Operational Practices", RFC 4641, September 2006.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6071] Frankel, S. and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap", RFC 6071, February 2011.

[RFC6234] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.

[RFC6376] Crocker, D., Hansen, T., and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures", RFC 6376, September 2011.

[RFC6394] Barnes, R., "Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE)", RFC 6394, October 2011.

[X.690] "Recommendation ITU-T X.690 (2002) | ISO/IEC 8825-1:2002, Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", 2002.

Appendix A. Operational Considerations for Deploying TLSA Records

A.1. Creating TLSA Records

When creating TLSA records, care must be taken to avoid misconfigurations. Section 4 of this document states that a TLSA RRset whose validation state is secure MUST be used. This means that the existence of such a RRset effectively disables other forms of name and path validation. A misconfigured TLSA RRset will effectively disable access to the TLS server for all conforming clients, and this document does not provide any means of making a gradual transition to using TLSA.

When creating TLSA records with certificate usage 0 (CA Certificate) or usage 2 (Trust Anchor), one needs to understand the implications when choosing between selector type 0 (full certificate) and 1 (SubjectPublicKeyInfo). A careful choice is required because different methods for building trust chains are used by different TLS clients. The following outlines the cases that one ought to be aware of and discusses the implications of the choice of selector type.

Certificate usage 2 is not affected by the different types of chain building when the end entity certificate is the same as the trust anchor certificate.

A.1.1. Ambiguities and Corner Cases When TLS Clients Build Trust Chains

TLS clients can implement their own chain-building code rather than rely on the chain presented by the TLS server. This means that, except for the end entity certificate, any certificate presented in the suggested chain might or might not be present in the final chain built by the client.

Certificates that the client can use to replace certificates from the original chain include:

- o Client's trust anchors
- o Certificates cached locally
- o Certificates retrieved from a URI listed in an Authority Information Access X.509v3 extension

CAs frequently reissue certificates with different validity periods, signature algorithms (such as an different hash algorithm in the signature algorithm), CA key pairs (such as for a cross-certificate), or PKIX extensions where the public key and subject remain the same. These reissued certificates are the certificates TLS client can use in place of an original certificate.

Clients are known to exchange or remove certificates that could cause TLSA certificate associations that rely on the full certificate to fail. For example:

- o The client considers the signature algorithm of a certificate to no longer be sufficiently secure
- o The client might not have an associated root certificate in its trust store and instead uses a cross-certificate with an identical subject and public key.

A.1.2. Choosing a Selector Type

In this section, "false-negative failure" means that a client will not accept the TLSA certificate association for a certificate designated by DNS administrator. Also, "false-positive acceptance" means that the client accepts a TLSA association for a certificate that is not designated by the DNS administrator.

A.1.2.1. Selector Type 0 (Full Certificate)

The "Full certificate" selector provides the most precise specification of a TLSA certificate association, capturing all fields of the PKIX certificate. For a DNS administrator, the best course to avoid false-negative failures in the client when using this selector is:

1. If a CA issued a replacement certificate, don't associate to CA certificates that have a signature algorithm with a hash that is considered weak by local policy.

2. Determine how common client applications process the TLSA certificate association using a fresh client installation, that is, with the local certificate cache empty.

A.1.2.2. Selector Type 1 (SubjectPublicKeyInfo)

A SubjectPublicKeyInfo selector gives greater flexibility in avoiding some false-negative failures caused by trust-chain-building algorithms used in clients.

One specific use-case ought to be noted: creating a TLSA certificate association to CA certificate I1 that directly signed end entity certificate S1 of the server. The case can be illustrated by following graph:

```
        +----+                      +----+
        | I1 |                      | I2 |
        +----+                      +----+
           |                           |
           v                           v
        +----+                      +----+
        | S1 |                      | S1 |
        +----+                      +----+
Certificate chain sent by    A different validation path
server in TLS handshake      built by the TLS client
```

I2 is a reissued version of CA certificate I1 (that is, it has a different hash in its signature algorithm).

In the above scenario, both certificates I1 and I2 that sign S1 need to have identical SubjectPublicKeyInfos because the key used to sign S1 is fixed. An association to SubjectPublicKeyInfo (selector type 1) will always succeed in such a case, but an association with a full certificate (selector type 0) might not work due to a false-negative failure.

The attack surface is a bit broader compared to "full certificate" selector: the DNS administrator might unintentionally specify an association that would lead to false-positive acceptance.

- o The administrator must know or trust that the CA does not engage in bad practices, such as not sharing the key of I1 for unrelated CA certificates leading to trust-chain redirect. If possible, the administrator ought to review all CA certificates that have the same SPKI.

- o The administrator ought to understand whether some PKIX extension may adversely affect security of the association. If possible, administrators ought to review all CA certificates that share the SubjectPublicKeyInfo.

- The administrator ought to understand that any CA could, in the future, issue a certificate that contains the same SubjectPublicKeyInfo. Therefore, new chains can crop up in the future without any warning.

Using the SubjectPublicKeyInfo selector for association with a certificate in a chain above I1 needs to be decided on a case-by-case basis: there are too many possibilities based on the issuing CA's practices. Unless the full implications of such an association are understood by the administrator, using selector type 0 is a better option from a security perspective.

A.2. Provisioning TLSA Records in DNS

A.2.1. Provisioning TLSA Records with Aliases

The TLSA resource record is not special in the DNS; it acts exactly like any other RRtype where the queried name has one or more labels prefixed to the base name, such as the SRV RRtype [RFC2782]. This affects the way that the TLSA resource record is used when aliasing in the DNS.

Note that the IETF sometimes adds new types of aliasing in the DNS. If that happens in the future, those aliases might affect TLSA records, hopefully in a good way.

A.2.1.1. Provisioning TLSA Records with CNAME Records

Using CNAME to alias in DNS only aliases from the exact name given, not any zones below the given name. For example, assume that a zone file has only the following:

```
sub1.example.com.          IN CNAME sub2.example.com.
```

In this case, a request for the A record at "bottom.sub1.example.com" would not return any records because the CNAME given only aliases the name given. Assume, instead, the zone file has the following:

```
sub3.example.com.          IN CNAME sub4.example.com.
bottom.sub3.example.com.   IN CNAME bottom.sub4.example.com.
```

In this case, a request for the A record at bottom.sub3.example.com would in fact return whatever value for the A record exists at bottom.sub4.example.com.

Application implementations and full-service resolvers request DNS records using libraries that automatically follow CNAME (and DNAME) aliasing. This allows hosts to put TLSA records in their own zones

or to use CNAME to do redirection.

If the owner of the original domain wants a TLSA record for the same, they simply enter it under the defined prefix:

```
; No TLSA record in target domain
;
sub5.example.com.            IN CNAME sub6.example.com.
_443._tcp.sub5.example.com.  IN TLSA 1 1 1 308202c5308201ab...
sub6.example.com.            IN A 192.0.2.1
sub6.example.com.            IN AAAA 2001:db8::1
```

If the owner of the original domain wants to have the target domain host the TLSA record, the original domain uses a CNAME record:

```
; TLSA record for original domain has CNAME to target domain
;
sub5.example.com.            IN CNAME sub6.example.com.
_443._tcp.sub5.example.com.  IN CNAME _443._tcp.sub6.example.com.
sub6.example.com.            IN A 192.0.2.1
sub6.example.com.            IN AAAA 2001:db8::1
_443._tcp.sub6.example.com.  IN TLSA 1 1 1 536a570ac49d9ba4...
```

Note that it is acceptable for both the original domain and the target domain to have TLSA records, but the two records are unrelated. Consider the following:

```
; TLSA record in both the original and target domain
;
sub5.example.com.            IN CNAME sub6.example.com.
_443._tcp.sub5.example.com.  IN TLSA 1 1 1 308202c5308201ab...
sub6.example.com.            IN A 192.0.2.1
sub6.example.com.            IN AAAA 2001:db8::1
_443._tcp.sub6.example.com.  IN TLSA 1 1 1 ac49d9ba4570ac49...
```

In this example, someone looking for the TLSA record for sub5.example.com would always get the record whose value starts "308202c5308201ab"; the TLSA record whose value starts "ac49d9ba4570ac49" would only be sought by someone who is looking for the TLSA record for sub6.example.com, and never for sub5.example.com. Note that deploying different certificates for multiple services located at a shared TLS listener often requires the use of TLS SNI (Server Name Indication) [RFC6066].

Note that these methods use the normal method for DNS aliasing using CNAME: the DNS client requests the record type that they actually want.

A.2.1.2. Provisioning TLSA Records with DNAME Records

Using DNAME records allows a zone owner to alias an entire subtree of names below the name that has the DNAME. This allows the wholesale aliasing of prefixed records such as those used by TLSA, SRV, and so on without aliasing the name itself. However, because DNAME can only be used for subtrees of a base name, it is rarely used to alias individual hosts that might also be running TLS.

```
; TLSA record in target domain, visible in original domain via DNAME
;
sub5.example.com.            IN CNAME sub6.example.com.
_tcp.sub5.example.com.       IN DNAME _tcp.sub6.example.com.
sub6.example.com.            IN A 192.0.2.1
sub6.example.com.            IN AAAA 2001:db8::1
_443._tcp.sub6.example.com.  IN TLSA 1 1 1 536a570ac49d9ba4...
```

A.2.1.3. Provisioning TLSA Records with Wildcards

Wildcards are generally not terribly useful for RRtypes that require prefixing because one can only wildcard at a layer below the host name. For example, if one wants to have the same TLSA record for every TCP port for www.example.com, the result might be:

```
*._tcp.www.example.com.    IN TLSA 1 1 1 5c1502a6549c423b...
```

This is possibly useful in some scenarios where the same service is offered on many ports or the same certificate and/or key is used for all services on a host. Note that the domain being searched for is not necessarily related to the domain name found in the certificate, so a certificate with a wildcard in it is not searched for using a wildcard in the search request.

A.3. Securing the Last Hop

As described in Section 4, an application processing TLSA records must know the DNSSEC validity of those records. There are many ways for the application to determine this securely, and this specification does not mandate any single method.

Some common methods for an application to know the DNSSEC validity of TLSA records include:

- o The application can have its own DNS resolver and DNSSEC validation stack.

- o The application can communicate through a trusted channel (such as requests to the operating system under which the application is

running) to a local DNS resolver that does DNSSEC validation.

- The application can communicate through a secured channel (such as requests running over TLS, IPsec, TSIG or SIG(0)) to a non-local DNS resolver that does DNSSEC validation.

- The application can communicate through a secured channel (such as requests running over TLS, IPsec, TSIG or SIG(0)) to a non-local DNS resolver that does not do DNSSEC validation, but gets responses through a secured channel from a different DNS resolver that does DNSSEC validation.

A.4. Handling Certificate Rollover

Certificate rollover is handled in much the same was as for rolling DNSSEC zone signing keys using the pre-publish key rollover method [RFC4641]. Suppose example.com has a single TLSA record for a TLS service on TCP port 990:

```
_990._tcp.example.com IN TLSA 1 1 1 1CFC98A706BCF3683015...
```

To start the rollover process, obtain or generate the new certificate or SubjectPublicKeyInfo to be used after the rollover and generate the new TLSA record. Add that record alongside the old one:

```
_990._tcp.example.com IN TLSA 1 1 1 1CFC98A706BCF3683015...
_990._tcp.example.com IN TLSA 1 1 1 62D5414CD1CC657E3D30...
```

After the new records have propagated to the authoritative nameservers and the TTL of the old record has expired, switch to the new certificate on the TLS server. Once this has occurred, the old TLSA record can be removed:

```
_990._tcp.example.com IN TLSA 1 1 1 62D5414CD1CC657E3D30...
```

This completes the certificate rollover.

Appendix B. Pseudocode for Using TLSA

This appendix describes the interactions given earlier in this specification in pseudocode format. If the steps below disagree with the text earlier in the document, the steps earlier in the document ought to be considered correct and this text incorrect.

Note that this pseudocode is more strict than the normative text. For instance, it forces an order on the evaluation of criteria which is not mandatory from the normative text.

B.1. Helper Functions

```
// implement the function for exiting
function Finish (F) = {
  if (F == ABORT_TLS) {
    abort the TLS handshake or prevent TLS from starting
    exit
  }

  if (F == NO_TLSA) {
    fall back to non-TLSA certificate validation
    exit
  }

  if (F == ACCEPT) {
    accept the TLS connection
    exit
  }

  // unreachable
}

// implement the selector function
function Select (S, X) = {
  // Full certificate
  if (S == 0) {
    return X in DER encoding
  }

  // SubjectPublicKeyInfo
  if (S == 1) {
    return X.SubjectPublicKeyInfo in DER encoding
  }

  // unreachable
}

// implement the matching function
function Match (M, X, Y) {
  // Exact match on selected content
  if (M == 0) {
    return (X == Y)
  }

  // SHA-256 hash of selected content
  if (M == 1) {
    return (SHA-256(X) == Y)
```

```
     }

     // SHA-512 hash of selected content
     if (M == 2) {
       return (SHA-512(X) == Y)
     }

     // unreachable
   }
```

B.2. Main TLSA Pseudo Code

TLS connect using [transport] to [name] on [port] and receiving end entity cert C for the TLS server:

```
   (TLSArecords, ValState) = DNSSECValidatedLookup(
     domainname=_[port]._[transport].[name], RRtype=TLSA)

   // check for states that would change processing
   if (ValState == BOGUS) {
     Finish(ABORT_TLS)
   }
   if ((ValState == INDETERMINATE) or (ValState == INSECURE)) {
     Finish(NO_TLSA)
   }
   // if here, ValState must be SECURE

   for each R in TLSArecords {
     // unusable records include unknown certUsage, unknown
     // selectorType, unknown matchingType, erroneous RDATA, and
     // prohibited by local policy
     if (R is unusable) {
       remove R from TLSArecords
     }
   }
   if (length(TLSArecords) == 0) {
     Finish(NO_TLSA)
   }

   // A TLS client might have multiple trust anchors that it might use
   //    when validating the TLS server's end entity certificate. Also,
   //    there can be multiple PKIX certification paths for the
   //    certificates given by the server in TLS. Thus, there are
   //    possibly many chains that might need to be tested during
   //    PKIX path validation.
```

```
   for each R in TLSArecords {

     // pass PKIX certificate validation and chain through a CA cert
     //    that comes from TLSA
     if (R.certUsage == 0) {
       for each PKIX certification path H {
         if (C passes PKIX certification path validation in H) {
           for each D in H {
             if ((D is a CA certificate) and
                 Match(R.matchingType, Select(R.selectorType, D),
                       R.cert)) {
               Finish(ACCEPT)
             }
           }
         }
       }
     }

     // pass PKIX certificate validation and match EE cert from TLSA
     if (R.certUsage == 1) {
       for each PKIX certification path H {
         if ((C passes PKIX certificate validation in H) and
                 Match(R.matchingType, Select(R.selectorType, C),
                 R.cert)) {
             Finish(ACCEPT)
         }
       }
     }

     // pass PKIX certification validation using TLSA record as the
     //    trust anchor
     if (R.certUsage == 2) {
       // the following assert() is merely a formalization of the
       // "trust anchor" condition for a certificate D matching R
       assert(Match(R.matchingType, Select(R.selectorType, D), R.cert))

       for each PKIX certification path H that has certificate D
           matching R as the trust anchor {
         if (C passes PKIX validation in H) {
           Finish(ACCEPT);
         }
       }
     }

     // match the TLSA record and the TLS certificate
     if (R.certUsage == 3) {
       if Match(R.matchingType, Select(R.selectorType, C), R.cert)
         Finish(ACCEPT)
```

```
       }
     }

   }

   // if here, then none of the TLSA records ended in "Finish(ACCEPT)"
   //   so abort TLS
   Finish(ABORT_TLS)
```

Appendix C. Examples

The following are examples of self-signed certificates that have been been generated with various selectors and matching types. They were generated with one piece of software, and validated by an individual using other tools.

```
S = Selector
M = Matching Type

S M Association Data
0 0 30820454308202BC020900AB58D24E77AD2AF6300D06092A86
    4886F70D0101050500306C310B30090603550406130245E4C31163014
    0603550408130D4E6F6F72642D486F6C6C616E6431123010060355504
    071309416D7374657264616D310C300A060355040A13034F53333123
    30210603550403131A64616E652E6B6965762E70726163746963756D
    2E6F73332E6E6C301E170D3132303131363136353730335A170D3232
    303131333136353730335A306C310B3009060355040613024E4C3116
    30140603550408130D4E6F6F72642D486F6C6C616E64311230100603
    5504071309416D7374657264616D310C300A060355040A13034F5333
    31233021060355040313 1A64616E652E6B6965762E70726163746963
    756D2E6F73332E6E6C308201A2300D06092A864886F70D0101010500
    0382018F003082018A0282018100E62C84A5AFE59F0A2A6B250DEE68
    7AC8C5C604F57D26CEB2119140FFAC38C4B9CBBE8923082E7F81626B
    6AD5DEA0C8771C74E3CAA7F613054AEFA3673E48FFE47B3F7AF987DE
    281A68230B24B9DA1A98DCBE51195B60E42FD7517C328D983E26A827
    C877AB914EE4C1BFDEAD48BD25BE5F2C473BA9C1CBBDDDA0C374D0D5
    8C389CC3D6D8C20662E19CF768F32441B7F7D14AEA8966CE7C32A172
    2AB38623D008029A9E4702883F8B977A1A1E5292BF8AD72239D40393
    37B86A3AC60FA001290452177BF1798609A05A130F033457A5212629
    FBDDB8E70E2A9E6556873C4F7CA46AE4A8B178F05FB319005E1C1C7D
    4BD77DFA34035563C126AA2C3328B900E7990AC9787F01DA82F74C3D
    4B6674CCECE1FD4C6EF9E6644F4635EDEDA39D8B0E2F7C8E06DAE775
    6213BD3D60831175BE290442B4AFC5AE6F46B769855A067C1097E617
    962529E166F22AEE10DDB981B8CD6FF17D3D70723169038DBFBC1A44
    9C8D0D31BC683C5F3CE26148E42EC9BBD4D9F261569B25B53C1D7FC2
    DDFF6B4CAC050203010001300D06092A864886F70D01010505000382
```

```
        0181002B2ABE063E9C86AC4A1F7835372091079C8276A9C2C5D1EC57
        64DE523FDDABDEAB3FD34E6FE6CBA054580A6785A663595D90132B93
        D473929E81FA0887D2FFF78A81C7D014B97778AB6AC9E5E690F6F5A9
        E92BB5FBAB71B857AE69B6E18BDCCB0BA6FCD9D4B084A34F3635148C
        495D48FE635903B888EC1DEB2610548EDD48D63F86513A4562469831
        48C0D5DB82D73A4C350A42BB661D763430FC6C8E5F9D13EA1B76AA52
        A4C358E5EA04000F794618303AB6CEEA4E9A8E9C74D73C1B0B7BAF16
        DEDE7696B5E2F206F777100F5727E1684D4132F5E692F47AF6756EA8
        B421000BE031B5D8F0220E436B51FB154FE9595333C13A2403F9DE08
        E5DDC5A22FD6182E339593E26374450220BC14F3E40FF33F084526B0
        9C34250702E8A352B332CCCB0F9DE2CF2B338823B92AFC61C0B6B8AB
        DB5AF718ED8DDA97C298E46B82A01B14814868CFA4F2C36268BFFF4A
        591F42658BF75918902D3E426DFE1D5FF0FC6A212071F6DA8BD833FE
        2E560D87775E8EE9333C05B6FB8EB56589D910DB5EA903

    0 1 EFDDF0D915C7BDC5782C0881E1B2A95AD099FBDD06D7B1F779
        82D9364338D955

    0 2 81EE7F6C0ECC6B09B7785A9418F54432DE630DD54DC6EE9E3C
        49DE547708D236D4C413C3E97E44F969E635958AA410495844127C04
        883503E5B024CF7A8F6A94

    1 0 308201A2300D06092A864886F70D01010105000382018F0030
        82018A0282018100E62C84A5AFE59F0A2A6B250DEE687AC8C5C604F5
        7D26CEB2119140FFAC38C4B9CBBE8923082E7F81626B6AD5DEA0C877
        1C74E3CAA7F613054AEFA3673E48FFE47B3F7AF987DE281A68230B24
        B9DA1A98DCBE51195B60E42FD7517C328D983E26A827C877AB914EE4
        C1BFDEAD48BD25BE5F2C473BA9C1CBBDDDA0C374D0D58C389CC3D6D8
        C20662E19CF768F32441B7F7D14AEA8966CE7C32A1722AB38623D008
        029A9E4702883F8B977A1A1E5292BF8AD72239D4039337B86A3AC60F
        A001290452177BF1798609A05A130F033457A5212629FBDDB8E70E2A
        9E6556873C4F7CA46AE4A8B178F05FB319005E1C1C7D4BD77DFA3403
        5563C126AA2C3328B900E7990AC9787F01DA82F74C3D4B6674CCECE1
        FD4C6EF9E6644F4635EDEDA39D8B0E2F7C8E06DAE7756213BD3D6083
        1175BE290442B4AFC5AE6F46B769855A067C1097E617962529E166F2
        2AEE10DDB981B8CD6FF17D3D70723169038DBFBC1A449C8D0D31BC68
        3C5F3CE26148E42EC9BBD4D9F261569B25B53C1D7FC2DDFF6B4CAC05
        0203010001

    1 1 8755CDAA8FE24EF16CC0F2C918063185E433FAAF1415664911
        D9E30A924138C4

    1 2 D43165B4CDF8F8660AECCCC5344D9D9AE45FFD7E6AAB7AB9EE
        C169B58E11F227ED90C17330CC17B5CCEF0390066008C720CEC6AAE5
        33A934B3A2D7E232C94AB4
```

Authors' Addresses

Paul Hoffman
VPN Consortium

Email: paul.hoffman@vpnc.org

Jakob Schlyter
Kirei AB

Email: jakob@kirei.se