Internet Engineering Task Force (IETF)
Request for Comments: 6066
Obsoletes: 4366
Category: Standards Track
ISSN: 2070-1721

D. Eastlake 3rd
Huawei

January 2011

# Transport Layer Security (TLS) Extensions: Extension Definitions

## Abstract

This document provides specifications for existing TLS extensions. It is a companion document for RFC 5246, "The Transport Layer Security (TLS) Protocol Version 1.2". The extensions specified are server_name, max_fragment_length, client_certificate_url, trusted_ca_keys, truncated_hmac, and status_request.

## Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6066.

Table of Contents

## 1. Introduction

The Transport Layer Security (TLS) Protocol Version 1.2 is specified in [RFC5246]. That specification includes the framework for extensions to TLS, considerations in designing such extensions (see Section 7.4.1.4 of [RFC5246]), and IANA Considerations for the allocation of new extension code points; however, it does not specify any particular extensions other than Signature Algorithms (see Section 7.4.1.4.1 of [RFC5246]).

This document provides the specifications for existing TLS extensions. It is, for the most part, the adaptation and editing of material from RFC 4366, which covered TLS extensions for TLS 1.0 (RFC 2246) and TLS 1.1 (RFC 4346).

### 1.1. Specific Extensions Covered

The extensions described here focus on extending the functionality provided by the TLS protocol message formats. Other issues, such as the addition of new cipher suites, are deferred.

The extension types defined in this document are:

```
enum {
    server_name(0), max_fragment_length(1),
    client_certificate_url(2), trusted_ca_keys(3),
    truncated_hmac(4), status_request(5), (65535)
} ExtensionType;
```

Specifically, the extensions described in this document:

- Allow TLS clients to provide to the TLS server the name of the server they are contacting. This functionality is desirable in order to facilitate secure connections to servers that host multiple 'virtual' servers at a single underlying network address.

- Allow TLS clients and servers to negotiate the maximum fragment length to be sent. This functionality is desirable as a result of memory constraints among some clients, and bandwidth constraints among some access networks.

- Allow TLS clients and servers to negotiate the use of client certificate URLs. This functionality is desirable in order to conserve memory on constrained clients.

- Allow TLS clients to indicate to TLS servers which certification authority (CA) root keys they possess. This functionality is desirable in order to prevent multiple handshake failures involving TLS clients that are only able to store a small number of CA root keys due to memory limitations.

- Allow TLS clients and servers to negotiate the use of truncated Message Authentication Codes (MACs). This functionality is desirable in order to conserve bandwidth in constrained access networks.

- Allow TLS clients and servers to negotiate that the server sends the client certificate status information (e.g., an Online Certificate Status Protocol (OCSP) [RFC2560] response) during a TLS handshake. This functionality is desirable in order to avoid sending a Certificate Revocation List (CRL) over a constrained access network and therefore saving bandwidth.

TLS clients and servers may use the extensions described in this document. The extensions are designed to be backwards compatible, meaning that TLS clients that support the extensions can talk to TLS servers that do not support the extensions, and vice versa.

Note that any messages associated with these extensions that are sent during the TLS handshake MUST be included in the hash calculations involved in "Finished" messages.

Note also that all the extensions defined in this document are relevant only when a session is initiated. A client that requests session resumption does not in general know whether the server will accept this request, and therefore it SHOULD send the same extensions as it would send if it were not attempting resumption. When a client includes one or more of the defined extension types in an extended client hello while requesting session resumption:

- The server name indication extension MAY be used by the server when deciding whether or not to resume a session as described in Section 3.

- If the resumption request is denied, the use of the extensions is negotiated as normal.

- If, on the other hand, the older session is resumed, then the server MUST ignore the extensions and send a server hello containing none of the extension types. In this case, the functionality of these extensions negotiated during the original session initiation is applied to the resumed session.

### 1.2. Conventions Used in This Document

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 [RFC2119].

## 2. Extensions to the Handshake Protocol

This document specifies the use of two new handshake messages, "CertificateURL" and "CertificateStatus". These messages are described in Sections 5 and 8, respectively. The new handshake message structure therefore becomes:

```
enum {
    hello_request(0), client_hello(1), server_hello(2),
    certificate(11), server_key_exchange (12),
    certificate_request(13), server_hello_done(14),
    certificate_verify(15), client_key_exchange(16),
    finished(20), certificate_url(21), certificate_status(22),
    (255)
} HandshakeType;

struct {
    HandshakeType msg_type;    /* handshake type */
    uint24 length;             /* bytes in message */
    select (HandshakeType) {
        case hello_request:       HelloRequest;
        case client_hello:        ClientHello;
        case server_hello:        ServerHello;
        case certificate:         Certificate;
        case server_key_exchange: ServerKeyExchange;
        case certificate_request: CertificateRequest;
        case server_hello_done:   ServerHelloDone;
        case certificate_verify:  CertificateVerify;
        case client_key_exchange: ClientKeyExchange;
        case finished:            Finished;
        case certificate_url:     CertificateURL;
        case certificate_status:  CertificateStatus;
    } body;
} Handshake;
```

## 3. Server Name Indication

TLS does not provide a mechanism for a client to tell a server the name of the server it is contacting. It may be desirable for clients to provide this information to facilitate secure connections to servers that host multiple 'virtual' servers at a single underlying network address.

In order to provide any of the server names, clients MAY include an extension of type "server_name" in the (extended) client hello. The "extension_data" field of this extension SHALL contain "ServerNameList" where:

```
   struct {
       NameType name_type;
       select (name_type) {
           case host_name: HostName;
       } name;
   } ServerName;

   enum {
       host_name(0), (255)
   } NameType;

   opaque HostName<1..2^16-1>;

   struct {
       ServerName server_name_list<1..2^16-1>
   } ServerNameList;
```

The ServerNameList MUST NOT contain more than one name of the same name_type. If the server understood the ClientHello extension but does not recognize the server name, the server SHOULD take one of two actions: either abort the handshake by sending a fatal-level unrecognized_name(112) alert or continue the handshake. It is NOT RECOMMENDED to send a warning-level unrecognized_name(112) alert, because the client's behavior in response to warning-level alerts is unpredictable. If there is a mismatch between the server name used by the client application and the server name of the credential chosen by the server, this mismatch will become apparent when the client application performs the server endpoint identification, at which point the client application will have to decide whether to proceed with the communication. TLS implementations are encouraged to make information available to application callers about warning-level alerts that were received or sent during a TLS handshake. Such information can be useful for diagnostic purposes.

> Note: Earlier versions of this specification permitted multiple names of the same name_type. In practice, current client implementations only send one name, and the client cannot necessarily find out which name the server selected. Multiple names of the same name_type are therefore now prohibited.

Currently, the only server names supported are DNS hostnames; however, this does not imply any dependency of TLS on DNS, and other name types may be added in the future (by an RFC that updates this document). The data structure associated with the host_name NameType is a variable-length vector that begins with a 16-bit length. For backward compatibility, all future data structures associated with new NameTypes MUST begin with a 16-bit length field. TLS MAY treat provided server names as opaque data and pass the names and types to the application.

"HostName" contains the fully qualified DNS hostname of the server, as understood by the client. The hostname is represented as a byte string using ASCII encoding without a trailing dot. This allows the support of internationalized domain names through the use of A-labels defined in [RFC5890]. DNS hostnames are case-insensitive. The algorithm to compare hostnames is described in [RFC5890], Section 2.3.2.4.

Literal IPv4 and IPv6 addresses are not permitted in "HostName".

It is RECOMMENDED that clients include an extension of type "server_name" in the client hello whenever they locate a server by a supported name type.

A server that receives a client hello containing the "server_name" extension MAY use the information contained in the extension to guide its selection of an appropriate certificate to return to the client, and/or other aspects of security policy. In this event, the server SHALL include an extension of type "server_name" in the (extended) server hello. The "extension_data" field of this extension SHALL be empty.

When the server is deciding whether or not to accept a request to resume a session, the contents of a server_name extension MAY be used in the lookup of the session in the session cache. The client SHOULD include the same server_name extension in the session resumption request as it did in the full handshake that established the session. A server that implements this extension MUST NOT accept the request to resume the session if the server_name extension contains a different name. Instead, it proceeds with a full handshake to establish a new session. When resuming a session, the server MUST NOT include a server_name extension in the server hello.

If an application negotiates a server name using an application protocol and then upgrades to TLS, and if a server_name extension is sent, then the extension SHOULD contain the same name that was negotiated in the application protocol. If the server_name is established in the TLS session handshake, the client SHOULD NOT attempt to request a different server name at the application layer.

## 4. Maximum Fragment Length Negotiation

Without this extension, TLS specifies a fixed maximum plaintext fragment length of 2^14 bytes. It may be desirable for constrained clients to negotiate a smaller maximum fragment length due to memory limitations or bandwidth limitations.

In order to negotiate smaller maximum fragment lengths, clients MAY include an extension of type "max_fragment_length" in the (extended) client hello. The "extension_data" field of this extension SHALL contain:

```
enum{
    2^9(1), 2^10(2), 2^11(3), 2^12(4), (255)
} MaxFragmentLength;
```

whose value is the desired maximum fragment length. The allowed values for this field are: 2^9, 2^10, 2^11, and 2^12.

Servers that receive an extended client hello containing a "max_fragment_length" extension MAY accept the requested maximum fragment length by including an extension of type "max_fragment_length" in the (extended) server hello. The "extension_data" field of this extension SHALL contain a "MaxFragmentLength" whose value is the same as the requested maximum fragment length.

If a server receives a maximum fragment length negotiation request for a value other than the allowed values, it MUST abort the handshake with an "illegal_parameter" alert. Similarly, if a client receives a maximum fragment length negotiation response that differs from the length it requested, it MUST also abort the handshake with an "illegal_parameter" alert.

Once a maximum fragment length other than 2^14 has been successfully negotiated, the client and server MUST immediately begin fragmenting messages (including handshake messages) to ensure that no fragment larger than the negotiated length is sent. Note that TLS already requires clients and servers to support fragmentation of handshake messages.

The negotiated length applies for the duration of the session including session resumptions.

The negotiated length limits the input that the record layer may process without fragmentation (that is, the maximum value of TLSPlaintext.length; see [RFC5246], Section 6.2.1). Note that the output of the record layer may be larger. For example, if the negotiated length is 2^9=512, then, when using currently defined cipher suites (those defined in [RFC5246] and [RFC2712]) and null compression, the record-layer output can be at most 805 bytes: 5 bytes of headers, 512 bytes of application data, 256 bytes of padding, and 32 bytes of MAC. This means that in this event a TLS record-layer peer receiving a TLS record-layer message larger than 805 bytes MUST discard the message and send a "record_overflow" alert, without decrypting the message. When this extension is used with Datagram Transport Layer Security (DTLS), implementations SHOULD NOT generate record_overflow alerts unless the packet passes message authentication.

## 5. Client Certificate URLs

Without this extension, TLS specifies that when client authentication is performed, client certificates are sent by clients to servers during the TLS handshake. It may be desirable for constrained clients to send certificate URLs in place of certificates, so that they do not need to store their certificates and can therefore save memory.

In order to negotiate sending certificate URLs to a server, clients MAY include an extension of type "client_certificate_url" in the (extended) client hello. The "extension_data" field of this extension SHALL be empty.

(Note that it is necessary to negotiate the use of client certificate URLs in order to avoid "breaking" existing TLS servers.)

Servers that receive an extended client hello containing a "client_certificate_url" extension MAY indicate that they are willing to accept certificate URLs by including an extension of type "client_certificate_url" in the (extended) server hello. The "extension_data" field of this extension SHALL be empty.

After negotiation of the use of client certificate URLs has been successfully completed (by exchanging hellos including "client_certificate_url" extensions), clients MAY send a "CertificateURL" message in place of a "Certificate" message as follows (see also Section 2):

```
enum {
    individual_certs(0), pkipath(1), (255)
} CertChainType;

struct {
    CertChainType type;
    URLAndHash url_and_hash_list<1..2^16-1>;
} CertificateURL;

struct {
    opaque url<1..2^16-1>;
    unint8 padding;
    opaque SHA1Hash[20];
} URLAndHash;
```

Here, "url_and_hash_list" contains a sequence of URLs and hashes. Each "url" MUST be an absolute URI reference according to [RFC3986] that can be immediately used to fetch the certificate(s).

When X.509 certificates are used, there are two possibilities:

- If CertificateURL.type is "individual_certs", each URL refers to a single DER-encoded X.509v3 certificate, with the URL for the client's certificate first.

- If CertificateURL.type is "pkipath", the list contains a single URL referring to a DER-encoded certificate chain, using the type PkiPath described in Section 10.1.

When any other certificate format is used, the specification that describes use of that format in TLS should define the encoding format of certificates or certificate chains, and any constraint on their ordering.

The "padding" byte MUST be 0x01. It is present to make the structure backwards compatible.

The hash corresponding to each URL is the SHA-1 hash of the certificate or certificate chain (in the case of X.509 certificates, the DER-encoded certificate or the DER-encoded PkiPath).

Note that when a list of URLs for X.509 certificates is used, the ordering of URLs is the same as that used in the TLS Certificate message (see [RFC5246], Section 7.4.2), but opposite to the order in which certificates are encoded in PkiPath. In either case, the self-signed root certificate MAY be omitted from the chain, under the assumption that the server must already possess it in order to validate it.

Servers receiving "CertificateURL" SHALL attempt to retrieve the client's certificate chain from the URLs and then process the certificate chain as usual. A cached copy of the content of any URL in the chain MAY be used, provided that the SHA-1 hash matches the hash of the cached copy.

Servers that support this extension MUST support the 'http' URI scheme for certificate URLs and MAY support other schemes. Use of other schemes than 'http', 'https', or 'ftp' may create unexpected problems.

If the protocol used is HTTP, then the HTTP server can be configured to use the Cache-Control and Expires directives described in [RFC2616] to specify whether and for how long certificates or certificate chains should be cached.

The TLS server MUST NOT follow HTTP redirects when retrieving the certificates or certificate chain. The URLs used in this extension MUST NOT be chosen to depend on such redirects.

If the protocol used to retrieve certificates or certificate chains returns a MIME-formatted response (as HTTP does), then the following MIME Content-Types SHALL be used: when a single X.509v3 certificate is returned, the Content-Type is "application/pkix-cert" [RFC2585], and when a chain of X.509v3 certificates is returned, the Content-Type is "application/pkix-pkipath" (Section 10.1).

The server MUST check that the SHA-1 hash of the contents of the object retrieved from that URL (after decoding any MIME Content-Transfer-Encoding) matches the given hash. If any retrieved object does not have the correct SHA-1 hash, the server MUST abort the handshake with a bad_certificate_hash_value(114) alert. This alert is always fatal.

Clients may choose to send either "Certificate" or "CertificateURL" after successfully negotiating the option to send certificate URLs. The option to send a certificate is included to provide flexibility to clients possessing multiple certificates.

If a server is unable to obtain certificates in a given CertificateURL, it MUST send a fatal certificate_unobtainable(111) alert if it requires the certificates to complete the handshake. If the server does not require the certificates, then the server continues the handshake. The server MAY send a warning-level alert in this case. Clients receiving such an alert SHOULD log the alert and continue with the handshake if possible.

## 6. Trusted CA Indication

Constrained clients that, due to memory limitations, possess only a small number of CA root keys may wish to indicate to servers which root keys they possess, in order to avoid repeated handshake failures.

In order to indicate which CA root keys they possess, clients MAY include an extension of type "trusted_ca_keys" in the (extended) client hello. The "extension_data" field of this extension SHALL contain "TrustedAuthorities" where:

```
struct {
    TrustedAuthority trusted_authorities_list<0..2^16-1>;
} TrustedAuthorities;

struct {
    IdentifierType identifier_type;
    select (identifier_type) {
        case pre_agreed: struct {};
        case key_sha1_hash: SHA1Hash;
        case x509_name: DistinguishedName;
        case cert_sha1_hash: SHA1Hash;
    } identifier;
} TrustedAuthority;

enum {
    pre_agreed(0), key_sha1_hash(1), x509_name(2),
    cert_sha1_hash(3), (255)
} IdentifierType;

opaque DistinguishedName<1..2^16-1>;
```

Here, "TrustedAuthorities" provides a list of CA root key identifiers that the client possesses. Each CA root key is identified via either:

- "pre_agreed": no CA root key identity supplied.

- "key_sha1_hash": contains the SHA-1 hash of the CA root key. For Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) keys, this is the hash of the "subjectPublicKey" value. For RSA keys, the hash is of the big-endian byte string representation of the modulus without any initial zero-valued bytes. (This copies the key hash formats deployed in other environments.)

- "x509_name": contains the DER-encoded X.509 DistinguishedName of the CA.

- "cert_sha1_hash": contains the SHA-1 hash of a DER-encoded Certificate containing the CA root key.

Note that clients may include none, some, or all of the CA root keys they possess in this extension.

Note also that it is possible that a key hash or a Distinguished Name alone may not uniquely identify a certificate issuer (for example, if a particular CA has multiple key pairs). However, here we assume this is the case following the use of Distinguished Names to identify certificate issuers in TLS.

The option to include no CA root keys is included to allow the client to indicate possession of some pre-defined set of CA root keys.

Servers that receive a client hello containing the "trusted_ca_keys" extension MAY use the information contained in the extension to guide their selection of an appropriate certificate chain to return to the client. In this event, the server SHALL include an extension of type "trusted_ca_keys" in the (extended) server hello. The "extension_data" field of this extension SHALL be empty.

## 7. Truncated HMAC

Currently defined TLS cipher suites use the MAC construction HMAC [RFC2104] to authenticate record-layer communications. In TLS, the entire output of the hash function is used as the MAC tag. However, it may be desirable in constrained environments to save bandwidth by truncating the output of the hash function to 80 bits when forming MAC tags.

In order to negotiate the use of 80-bit truncated HMAC, clients MAY include an extension of type "truncated_hmac" in the extended client hello. The "extension_data" field of this extension SHALL be empty.

Servers that receive an extended hello containing a "truncated_hmac" extension MAY agree to use a truncated HMAC by including an extension of type "truncated_hmac", with empty "extension_data", in the extended server hello.

Note that if new cipher suites are added that do not use HMAC, and the session negotiates one of these cipher suites, this extension will have no effect. It is strongly recommended that any new cipher suites using other MACs consider the MAC size an integral part of the

cipher suite definition, taking into account both security and bandwidth considerations.

If HMAC truncation has been successfully negotiated during a TLS handshake, and the negotiated cipher suite uses HMAC, both the client and the server pass this fact to the TLS record layer along with the other negotiated security parameters. Subsequently during the session, clients and servers MUST use truncated HMACs, calculated as specified in [RFC2104]. That is, SecurityParameters.mac_length is 10 bytes, and only the first 10 bytes of the HMAC output are transmitted and checked. Note that this extension does not affect the calculation of the pseudo-random function (PRF) as part of handshaking or key derivation.

The negotiated HMAC truncation size applies for the duration of the session including session resumptions.

## 8. Certificate Status Request

Constrained clients may wish to use a certificate-status protocol such as OCSP [RFC2560] to check the validity of server certificates, in order to avoid transmission of CRLs and therefore save bandwidth on constrained networks. This extension allows for such information to be sent in the TLS handshake, saving roundtrips and resources.

In order to indicate their desire to receive certificate status information, clients MAY include an extension of type "status_request" in the (extended) client hello. The "extension_data" field of this extension SHALL contain "CertificateStatusRequest" where:

```
struct {
    CertificateStatusType status_type;
    select (status_type) {
        case ocsp: OCSPStatusRequest;
    } request;
} CertificateStatusRequest;

enum { ocsp(1), (255) } CertificateStatusType;

struct {
    ResponderID responder_id_list<0..2^16-1>;
    Extensions  request_extensions;
} OCSPStatusRequest;

opaque ResponderID<1..2^16-1>;
opaque Extensions<0..2^16-1>;
```

In the OCSPStatusRequest, the "ResponderIDs" provides a list of OCSP responders that the client trusts. A zero-length "responder_id_list" sequence has the special meaning that the responders are implicitly known to the server, e.g., by prior arrangement. "Extensions" is a DER encoding of OCSP request extensions.

Both "ResponderID" and "Extensions" are DER-encoded ASN.1 types as defined in [RFC2560]. "Extensions" is imported from [RFC5280]. A zero-length "request_extensions" value means that there are no extensions (as opposed to a zero-length ASN.1 SEQUENCE, which is not valid for the "Extensions" type).

In the case of the "id-pkix-ocsp-nonce" OCSP extension, [RFC2560] is unclear about its encoding; for clarification, the nonce MUST be a DER-encoded OCTET STRING, which is encapsulated as another OCTET STRING (note that implementations based on an existing OCSP client will need to be checked for conformance to this requirement).

Servers that receive a client hello containing the "status_request" extension MAY return a suitable certificate status response to the client along with their certificate. If OCSP is requested, they SHOULD use the information contained in the extension when selecting an OCSP responder and SHOULD include request_extensions in the OCSP request.

Servers return a certificate response along with their certificate by sending a "CertificateStatus" message immediately after the "Certificate" message (and before any "ServerKeyExchange" or "CertificateRequest" messages). If a server returns a "CertificateStatus" message, then the server MUST have included an extension of type "status_request" with empty "extension_data" in the extended server hello. The "CertificateStatus" message is conveyed using the handshake message type "certificate_status" as follows (see also Section 2):

```
struct {
    CertificateStatusType status_type;
    select (status_type) {
        case ocsp: OCSPResponse;
    } response;
} CertificateStatus;

opaque OCSPResponse<1..2^24-1>;
```

An "ocsp_response" contains a complete, DER-encoded OCSP response (using the ASN.1 type OCSPResponse defined in [RFC2560]). Only one OCSP response may be sent.

Note that a server MAY also choose not to send a "CertificateStatus" message, even if has received a "status_request" extension in the client hello message and has sent a "status_request" extension in the server hello message.

Note in addition that a server MUST NOT send the "CertificateStatus" message unless it received a "status_request" extension in the client hello message and sent a "status_request" extension in the server hello message.

Clients requesting an OCSP response and receiving an OCSP response in a "CertificateStatus" message MUST check the OCSP response and abort the handshake if the response is not satisfactory with bad_certificate_status_response(113) alert. This alert is always fatal.

## 9. Error Alerts

Four new error alerts are defined for use with the TLS extensions defined in this document. To avoid "breaking" existing clients and servers, these alerts MUST NOT be sent unless the sending party has received an extended hello message from the party they are communicating with. These error alerts are conveyed using the following syntax. The new alerts are the last four, as indicated by the comments on the same line as the error alert number.

```
   enum {
       close_notify(0),
       unexpected_message(10),
       bad_record_mac(20),
       decryption_failed(21),
       record_overflow(22),
       decompression_failure(30),
       handshake_failure(40),
       /* 41 is not defined, for historical reasons */
       bad_certificate(42),
       unsupported_certificate(43),
       certificate_revoked(44),
       certificate_expired(45),
       certificate_unknown(46),
       illegal_parameter(47),
       unknown_ca(48),
       access_denied(49),
       decode_error(50),
       decrypt_error(51),
       export_restriction(60),
       protocol_version(70),
       insufficient_security(71),
```

```
          internal_error(80),
          user_canceled(90),
          no_renegotiation(100),
          unsupported_extension(110),
          certificate_unobtainable(111),        /* new */
          unrecognized_name(112),               /* new */
          bad_certificate_status_response(113), /* new */
          bad_certificate_hash_value(114),      /* new */
          (255)
      } AlertDescription;
```

"certificate_unobtainable" is described in Section 5.
"unrecognized_name" is described in Section 3.
"bad_certificate_status_response" is described in Section 8.
"bad_certificate_hash_value" is described in Section 5.

## 10. IANA Considerations

IANA Considerations for TLS extensions and the creation of a registry are covered in Section 12 of [RFC5246] except for the registration of MIME type application/pkix-pkipath, which appears below.

The IANA TLS extensions and MIME type application/pkix-pkipath registry entries that reference RFC 4366 have been updated to reference this document.

### 10.1. pkipath MIME Type Registration

MIME media type name: application
MIME subtype name: pkix-pkipath
Required parameters: none

Optional parameters: version (default value is "1")

Encoding considerations:
Binary; this MIME type is a DER encoding of the ASN.1 type PkiPath, defined as follows:

```
PkiPath ::= SEQUENCE OF Certificate
```

PkiPath is used to represent a certification path. Within the sequence, the order of certificates is such that the subject of the first certificate is the issuer of the second certificate, etc.

This is identical to the definition published in [X509-4th-TC1]; note that it is different from that in [X509-4th].

All Certificates MUST conform to [RFC5280]. (This should be interpreted as a requirement to encode only PKIX-conformant certificates using this type. It does not necessarily require

that all certificates that are not strictly PKIX-conformant must be rejected by relying parties, although the security consequences of accepting any such certificates should be considered carefully.)

DER (as opposed to BER) encoding MUST be used. If this type is sent over a 7-bit transport, base64 encoding SHOULD be used.

Security considerations:
The security considerations of [X509-4th] and [RFC5280] (or any updates to them) apply, as well as those of any protocol that uses this type (e.g., TLS).

Note that this type only specifies a certificate chain that can be assessed for validity according to the relying party's existing configuration of trusted CAs; it is not intended to be used to specify any change to that configuration.

Interoperability considerations:
No specific interoperability problems are known with this type, but for recommendations relating to X.509 certificates in general, see [RFC5280].

Published specification: This document and [RFC5280].

Applications that use this media type:
TLS. It may also be used by other protocols or for general interchange of PKIX certificate chains.

Additional information:
Magic number(s): DER-encoded ASN.1 can be easily recognized. Further parsing is required to distinguish it from other ASN.1 types.
File extension(s): .pkipath
Macintosh File Type Code(s): not specified

Person & email address to contact for further information:
Magnus Nystrom <mnystrom@microsoft.com>

Intended usage: COMMON

Change controller: IESG <iesg@ietf.org>

### 10.2. Reference for TLS Alerts, TLS HandshakeTypes, and ExtensionTypes

The following values in the TLS Alert Registry have been updated to reference this document:

111 certificate_unobtainable
112 unrecognized_name
113 bad_certificate_status_response
114 bad_certificate_hash_value

The following values in the TLS HandshakeType Registry have been updated to reference this document:

21 certificate_url
22 certificate_status

The following ExtensionType values have been updated to reference this document:

0 server_name
1 max_fragment_length
2 client_certificate_url
3 trusted_ca_keys
4 truncated_hmac
5 status_request

## 11. Security Considerations

General security considerations for TLS extensions are covered in [RFC5246]. Security Considerations for particular extensions specified in this document are given below.

In general, implementers should continue to monitor the state of the art and address any weaknesses identified.

### 11.1. Security Considerations for server_name

If a single server hosts several domains, then clearly it is necessary for the owners of each domain to ensure that this satisfies their security needs. Apart from this, server_name does not appear to introduce significant security issues.

Since it is possible for a client to present a different server_name in the application protocol, application server implementations that rely upon these names being the same MUST check to make sure the client did not present a different name in the application protocol.

Implementations MUST ensure that a buffer overflow does not occur, whatever the values of the length fields in server_name.

### 11.2. Security Considerations for max_fragment_length

The maximum fragment length takes effect immediately, including for handshake messages. However, that does not introduce any security complications that are not already present in TLS, since TLS requires implementations to be able to handle fragmented handshake messages.

Note that, as described in Section 4, once a non-null cipher suite has been activated, the effective maximum fragment length depends on the cipher suite and compression method, as well as on the negotiated max_fragment_length. This must be taken into account when sizing buffers and checking for buffer overflow.

### 11.3. Security Considerations for client_certificate_url

Support for client_certificate_url involves the server's acting as a client in another URI-scheme-dependent protocol. The server therefore becomes subject to many of the same security concerns that clients of the URI scheme are subject to, with the added concern that the client can attempt to prompt the server to connect to some (possibly weird-looking) URL.

In general, this issue means that an attacker might use the server to indirectly attack another host that is vulnerable to some security flaw. It also introduces the possibility of denial-of-service attacks in which an attacker makes many connections to the server, each of which results in the server's attempting a connection to the target of the attack.

Note that the server may be behind a firewall or otherwise able to access hosts that would not be directly accessible from the public Internet. This could exacerbate the potential security and denial-of-service problems described above, as well as allow the existence of internal hosts to be confirmed when they would otherwise be hidden.

The detailed security concerns involved will depend on the URI schemes supported by the server. In the case of HTTP, the concerns are similar to those that apply to a publicly accessible HTTP proxy server. In the case of HTTPS, loops and deadlocks may be created, and this should be addressed. In the case of FTP, attacks arise that are similar to FTP bounce attacks.

As a result of this issue, it is RECOMMENDED that the client_certificate_url extension should have to be specifically enabled by a server administrator, rather than be enabled by default. It is also RECOMMENDED that URI schemes be enabled by the administrator individually, and only a minimal set of schemes be enabled. Unusual protocols that offer limited security or whose security is not well understood SHOULD be avoided.

As discussed in [RFC3986], URLs that specify ports other than the default may cause problems, as may very long URLs (which are more likely to be useful in exploiting buffer overflow bugs).

This extension continues to use SHA-1 (as in RFC 4366) and does not provide algorithm agility. The property required of SHA-1 in this case is second pre-image resistance, not collision resistance. Furthermore, even if second pre-image attacks against SHA-1 are found in the future, an attack against client_certificate_url would require a second pre-image that is accepted as a valid certificate by the server and contains the same public key.

Also note that HTTP caching proxies are common on the Internet, and some proxies do not check for the latest version of an object correctly. If a request using HTTP (or another caching protocol) goes through a misconfigured or otherwise broken proxy, the proxy may return an out-of-date response.

### 11.4. Security Considerations for trusted_ca_keys

Potentially, the CA root keys a client possesses could be regarded as confidential information. As a result, the CA root key indication extension should be used with care.

The use of the SHA-1 certificate hash alternative ensures that each certificate is specified unambiguously. This context does not require a cryptographic hash function, so the use of SHA-1 is considered acceptable, and no algorithm agility is provided.

### 11.5. Security Considerations for truncated_hmac

It is possible that truncated MACs are weaker than "un-truncated" MACs. However, no significant weaknesses are currently known or expected to exist for HMAC with MD5 or SHA-1, truncated to 80 bits.

Note that the output length of a MAC need not be as long as the length of a symmetric cipher key, since forging of MAC values cannot be done off-line: in TLS, a single failed MAC guess will cause the immediate termination of the TLS session.

Since the MAC algorithm only takes effect after all handshake messages that affect extension parameters have been authenticated by the hashes in the Finished messages, it is not possible for an active attacker to force negotiation of the truncated HMAC extension where it would not otherwise be used (to the extent that the handshake authentication is secure). Therefore, in the event that any security problems were found with truncated HMAC in the future, if either the client or the server for a given session were updated to take the problem into account, it would be able to veto use of this extension.

### 11.6. Security Considerations for status_request

If a client requests an OCSP response, it must take into account that an attacker's server using a compromised key could (and probably would) pretend not to support the extension. In this case, a client that requires OCSP validation of certificates SHOULD either contact the OCSP server directly or abort the handshake.

Use of the OCSP nonce request extension (id-pkix-ocsp-nonce) may improve security against attacks that attempt to replay OCSP responses; see Section 4.4.1 of [RFC2560] for further details.

## 12. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

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

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, May 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 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.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

## 13. Informative References

[RFC2712] Medvinsky, A. and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712, October 1999.

[X509-4th] ITU-T Recommendation X.509 (2000) | ISO/IEC 9594-8:2001, "Information Systems - Open Systems Interconnection - The Directory: Public key and attribute certificate frameworks".

[X509-4th-TC1] ITU-T Recommendation X.509(2000) Corrigendum 1(2001) | ISO/IEC 9594-8:2001/Cor.1:2002, Technical Corrigendum 1 to ISO/IEC 9594:8:2001.

## Appendix A. Changes from RFC 4366

The significant changes between RFC 4366 and this document are described below.

RFC 4366 described both general extension mechanisms (for the TLS handshake and client and server hellos) as well as specific extensions. RFC 4366 was associated with RFC 4346, TLS 1.1. The client and server hello extension mechanisms have been moved into RFC 5246, TLS 1.2, so this document, which is associated with RFC 5246, includes only the handshake extension mechanisms and the specific extensions from RFC 4366. RFC 5246 also specifies the unknown extension error and new extension specification considerations, so that material has been removed from this document.

The Server Name extension now specifies only ASCII representation, eliminating UTF-8. It is provided that the ServerNameList can contain more than only one name of any particular name_type. If a server name is provided but not recognized, the server should either continue the handshake without an error or send a fatal error. Sending a warning-level message is not recommended because client behavior will be unpredictable. Provision was added for the user using the server_name extension in deciding whether or not to resume a session. Furthermore, this extension should be the same in a session resumption request as it was in the full handshake that established the session. Such a resumption request must not be accepted if the server_name extension is different, but instead a full handshake must be done to possibly establish a new session.

The Client Certificate URLs extension has been changed to make the presence of a hash mandatory.

For the case of DTLS, the requirement to report an overflow of the negotiated maximum fragment length is made conditional on passing authentication.

TLS servers are now prohibited from following HTTP redirects when retrieving certificates.

The material was also re-organized in minor ways. For example, information as to which errors are fatal is moved from the "Error Alerts" section to the individual extension specifications.

## Appendix B. Acknowledgements

This document is based on material from RFC 4366 for which the authors were S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Other contributors include Joseph Salowey, Alexey Melnikov, Peter Saint-Andre, and Adrian Farrel.

## Author's Address

Donald Eastlake 3rd
Huawei
155 Beaver Street
Milford, MA 01757 USA

Phone: +1-508-333-2270
EMail: d3e3e3@gmail.com