

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 16 June 2022

J. Henry
S. Orr
Cisco
13 December 2021

Fast Transition for Opportunistic Wireless Encryption (FT-OWE)
draft-henry-ft-owe-01

Abstract

Opportunistic Wireless Encryption, defined in RFC 8110, specifies an extension to the IEEE Std 802.11 to provide for opportunistic (unauthenticated) encryption to a specific wireless access point (AP). This memo extends the method to allow the establishment of OWE keying material to other APs before connection establishment to these APs, thus enabling a fast transition mode for OWE.

Status of This Memo

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

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

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

This Internet-Draft will expire on 16 June 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Requirements language	2
3. Background	2
3.1. Crowd Wisdom	3
3.2. 802.11 Fast Transition	4
4. FT-OWE Operations	4
4.1. Cryptography and Key Hierarchy	4
4.2. FT-OWE Discovery	6
4.3. FT-OWE Association	6
4.4. FT-OWE Post-Association and Roaming	6
4.4.1. Over-the-air FT-OWE authentication	7
4.4.2. Over-the-DS FT-OWE authentication	7
5. IANA Considerations	8
6. Security Considerations	8
Authors' Addresses	8

1. Introduction

This document describes an extension to Opportunistic Wireless Encryption (OWE), defined in [RFC8110], that is compatible with 802.11 multi-AP environments, where encrypted connections are expected between one client and more than one AP, with the light requirement that APs belong to the same Extended Service Set (ESS), i.e. communicate with one another over the same infrastructure.

2. Requirements language

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

3. Background

Opportunistic Wireless Encryption provides a mechanism for an 802.11 station (STA) to establish an encrypted connection with an 802.11 access point (AP). OWE does not provide authentication, which means that the STA cannot know if the AP is legitimate or not (i.e. is part of the local network infrastructure or belongs to an attacker). In many environments, the STA may need to establish an encrypted connection to multiple APs in succession and with short intervals. For example, the 802.11 Fine Timing Measurement (FTM) procedure supposes that a STA would measure its distance to multiple APs (which individual location is shared), and deduce from these measures its own location. To increase the privacy of the exchanges, it is

desirable that those exchanges would be protected from eavesdroppers. In such scenario, the STA would need to establish an encrypted link to each AP, in a rapid and consecutive manner. There is a need for a mechanism to facilitate the secure key exchange between the STA and these APs, without wasting time on each AP channels before ranging can start. In other scenarios, a STA would obtain information from one AP about other APs, for example through a Neighbor Report. The STA may then need to exchange information with these other APs, such as capabilities, data broadcast information, or more details about the supporting network through ANQP messages. To improve the reliability of the information obtained, the STA may want an indication that the second AP is part of the same system as the first AP. In such scenario, the STA would need to establish an encrypted link to the second AP, and obtain an indication that the second AP is likely part of the same system as the first AP. An extension of OWE allowing this multi-AP scenario provides such mechanism.

3.1. Crowd Wisdom

This requirement is especially present in public settings. A large venue accessible to the general public is likely to include multiple legitimate APs, that are all part of the same system (e.g., "mall Wi-Fi"). There may also be some APs that are legitimate, but that are not part of a larger system (e.g., small store individual AP). Last, there may be one or more illegitimate APs (e.g., attacker APs). Although OWE is not intended to provide authentication, extending OWE to a multiple-APs scenario also has the virtue of providing to the STA this indication that several APs can communicate with one another, and thus have established a trusted relationship over the network (e.g., through a wired communication, directly or via a central WLAN controller). Such communication is not a guarantee that the APs are legitimate, but offers a good indication that, if a first AP provides some information, the second AP is likely to provide information that is consistent of that of the first AP. As such, if FT-OWE does not provide authentication, it provides an element of crowd wisdom, where a STA can build confidence that a set of APs is coherent. This confidence is useful when a STA needs to exchange information with more than one AP in a short amount of time, and use that information to make a decision. For example, if APs 1 to 3 provide coherent location and ranging information and can communicate with one another, but APs 4 and 5 provide incoherent ranging or location information and cannot communicate with APs 1 to 3, then a location algorithm running on the end device would have no difficulty classifying APs 4 and 5 as suspicious outliers (e.g., possible evil twins), even if they announce the same SSIDs as APs 1 to 3, and even if APs 1 to 3 list APs 4 and 5 MAC addresses (BSSID) as known. Without the knowledge of such backend trusted communication, the client would be left with location values provided by five APs of

equal legitimacy value. The same logic applies to other unauthenticated exchanges, where the STA can easily form coherent groups of APs, which information is likely to display consistent value.

3.2. 802.11 Fast Transition

To extend OWE to a multi-AP scenario, a mechanism comparable to 802.11 FT is needed. 802.11 Fast Basic Service Set (BSS) Transition (FT) is a mechanism initially intended for associated and authenticated STAs and APs. With FT, a key hierarchy is created. A first level key (called Pairwise Master Key R0, PMK-R0) is used to generate second level sets of keys (PMK-R1s) that are each specific to the connection between a STA and a given AP. PMK-R0 is established when the STA connects to the network, and is used to derive the keying material specific to the connection of the STA to the first AP. Then, when the STA detects and needs to transition to a second AP, depending on what is signaled in the Mobility Domain Element from the AP [IEEE802.11] clause 9.4.2.46, the STA will authenticate Over-the-Air or Over-the-DS as defined in [IEEE802.11] section 13.5. With either method (Over-the-Air or Over-the-DS) the keying material (PMK-R1) required for the secured communication between the STA and the second AP is obtained before the STA joins the second AP. This process allows the STA to expedite the process of joining the second AP and resuming encrypted data exchange.

4. FT-OWE Operations

4.1. Cryptography and Key Hierarchy

As specified in [IEEE802.11] clause 12.7.6.1, the PMK is obtained upon successful authentication and association between the STA and the network. In a coordinated network system where APs communicate with one another, it is expected that a single entity (e.g., a Primary AP, or a Wireless LAN Controller) will be in charge of generating the public key defined in [RFC8110] section 4.3. Once the PMK generation between the STA and the network completes as per [RFC8110] section 4.4, a Master PMK is generated, following the process defined by [IEEE802.11] clause 12.7.1.6.3, where: MPMK = PMK generated as the result of OWE authentication in [RFC8110] section 4.4 PMKID = $\text{Truncate-128}(\text{Hash}(C \parallel A))$ as defined in [RFC8110] section 4.4, thus where C is the client's Diffie-Hellman public key from the 802.11 association request, A is the Authenticator (primary AP or WLAN controller) Diffie-Hellman public key from the 802.11 association response, and Hash is the hash algorithm defined in [RFC8110] section 4.1. Once the MPMK has been derived, each side (AP/WLC and STA) derives the PMK-R0 value, following [IEEE802.11] clause 12.7.1.6.3, and where: $\text{R0-Key-Data} = \text{KDF-Hash-Length}(\text{MPMK},$

$\backslash\text{FT-R0}$ ", SSIDlength || SSID || MDID || R0KHlength || R0KH-ID || S0KH-ID)

$\text{PMK-R0} = \text{L}(\text{R0-Key-Data}, 0, Q)$

$\text{PMK-R0Name-Salt} = \text{L}(\text{R0-Key-Data}, Q, 128)$

Length = $Q + 128$

Where Q is the length of the curve p defined in [RFC8110] section 4.1, KDF-Hash-Length is the key derivation function as defined in [IEEE802.11] clause 12.7.1.6.2 using the hash algorithm identified by [RFC8110] table 2,

SSIDlength is a single octet whose value is the number of octets in the SSID,

SSID is the service set identifier, a variable length sequence of octets, as it appears in the Beacon and Probe Response frames,

MDID is the Mobility Domain Identifier field from the Mobile Domain element (MDE) that was used during FT initial mobility domain association,

R0KHlength is a single octet whose value is the number of octets in the R0KH-ID,

R0KH-ID is the identifier of the holder of PMK-R0 in the Authenticator,

S0KH-ID is the STA, or Supplicant's MAC address (SPA).

The PMK-R0 is referenced and named as follows:

$\text{PMKR0Name} = \text{Truncate-128}(\text{Hash}(\backslash\text{FT-R0N} \parallel \text{PMK-R0Name-Salt}))$

where Hash is the hash algorithm identified by [RFC8110] table 2, $\backslash\text{FT-R0N}$ is treated as an ASCII string.

The PMKR0Name is used to identify the PMK-R0.

Once the PMK-R0 was determined, each side derives a PMK-R1 key, specific to the connection between the STA and a given AP, and used to derive the PTK. The PMK-R1 derivation follows the process defined in [IEEE802.11] clause 12.7.1.6.4, where:

$\text{PMK-R1} = \text{KDF-Hash-Length}(\text{PMK-R0}, \backslash\text{FT-R1N}, \text{R1KH-ID} \parallel \text{S1KH-ID})$

where KDF-Hash-Length is the key derivation function as defined in [IEEE802.11] 12.7.1.6.2,

Hash is the hash algorithm identified by [RFC8110] table 2,

Length is the length of the hash algorithm's digest,

PMK-R0 is the first level key in the FT key hierarchy,

R1KH-ID is a MAC address of the holder of the PMK-R1 in the Authenticator of the AP,

S1KH-ID is the SPA,

The PMK-R1 is then referenced and named as follows:

$\text{PMKR1Name} = \text{Truncate-128}(\text{Hash}(\backslash\text{FT-R1N} \parallel \text{PMKR0Name} \parallel \text{R1KH-ID} \parallel \text{S1KH-ID}))$ where Hash is the hash algorithm identified by [RFC8110] table 2,

$\backslash\text{FT-R1N}$ is treated as an ASCII string,

PMKR1Name is used to identify the PMK-R1.

The PTK is then defined following the KDF method described in [IEEE802.11] clause 12.7.1.6.5.

4.2. FT-OWE Discovery

An access point advertises support for FT-OWE using an Authentication and Key Management (AKM) suite selector for FT-OWE, illustrated in table 1, in its beacons and probe responses.

OUI	Suite	Authentication	Key	Key
Type	Type	Management	derivation	
Type	Type	type		
00-0F-AC	[TBD]	FT-Opportunistic	This	[IEEE802.11]
Wireless	Document	12.7.1.6.2		
Encryption				

The access point also advertises the Mobility Domain element (MDE) defined in [IEEE802.11] clause 9.4.2.46. The Mobility Domain element includes the 2-octets Mobility Domain Identifier that names the mobility domain supported by all APs in the same roaming domain, and the FT Capability and Policy Field that indicates if FT is set to occur over the air or over the DS. A STA in the process of discovering FT-OWE-compliant APs can use the above information, and can also insert the MDE in its probe requests.

4.3. FT-OWE Association

Once a STA discovers an FT-OWE-compliant AP, it performs an authentication as defined in [IEEE802.11], with the Authentication Algorithm number set to [TBD] (FT-OWE). The STA then proceeds to the FT-OWE association phase. In the Association Request frame, the STA includes the MDE defined in [IEEE802.11] and the Diffie-Hellman Parameter element (DHPE) defined in [RFC8110] section 4.3. The DHPE includes the STA public key. In the Association Response frame, the AP includes the MDE, and the Fast BSS Transition Element (FTE) defined in [IEEE802.11] clause 9.4.2.47. The FTE also includes the R0KH-ID and the R1KH-ID for the AP. The AP also includes the Diffie-Hellman Parameter element (DHPE) defined in [RFC8110] section 4.3. The DHPE includes the AP public key.

4.4. FT-OWE Post-Association and Roaming

Once association is completed, the STA and the AP derive the PMK-R0, then the PMK-R1 for the current STA/AP pair, then the matching PTK. Later, the STA will need to establish a secure association with another AP (called the target AP) part of the same mobility domain as a the current AP. In this scenario, it is expected that the STA will have discovered the target AP through a scanning process during which the target AP MAC address was discovered.

4.4.1. Over-the-air FT-OWE authentication

To perform OTA FT-OWE, the STA follows the process indicated in [IEEE802.11] clause 13.5.2. The STA first sends to the target AP an FT-OWE Authentication Request. The request indicates FT-OWE as the Authentication Algorithm, includes the RSNE with the PMKROName value, includes the MDE, and includes the FTE with a SNonce and the ROKH-ID. It is expected that the target AP should be able to communicate with a primary AP or a WLAN controller, recognize the PMKROName and ROKH-ID, and be able to derive the information needed to derive a PMKR1 value. The target AP responds with an FT-OWE Authentication Response. The response indicates FT-OWE as the Authentication Algorithm, includes the RSNE with the PMKROName value, includes the MDE, and includes the FTE with the ANonce, the SNonce, the target AP R1KH-ID and the ROKH-ID. At this stage, the FT-OWE authentication to the target AP has completed, and stays valid until the expiration of the reassociation deadline time. It is understood that the STA may establish such authentication to multiple target APs. Later, when the STA needs to associate to the target AP and proceed to secure exchanges, and while the authentication is still valid, the STA sends a FT-OWE reassociation request to the target AP. The request includes the RSNE with the PMKR1Name value, the MDE, the FTE with MIC (as defined in [RFC8110] section 4.4), ANonce, SNonce, the R1KH1-ID obtained during the authentication phase, ROKH-ID, and the Diffie-Hellman Parameter element (DHPE) defined in [RFC8110] section 4.3. The DHPE includes the STA public key. The Target AP responds with a FT-OWE Reassociation response. The response includes the RSNE with the PMKR1Name for the target AP, the MDE, the FTE with MIC (as defined in [RFC8110] section 4.4), ANonce, SNonce, the target AP R1KH1-ID, ROKH-ID, the and the Diffie-Hellman Parameter element (DHPE) defined in [RFC8110] section 4.3. The DHPE includes the STA public key. At the conclusion of the reassociation, both sides compute the PMKR1 as described in section 4.4, then the matching PTK.

4.4.2. Over-the-DS FT-OWE authentication

To perform Over-the-DS FT-OWE, the STA follows the process indicated in [IEEE802.11] clause 13.5.3. The STA first sends to the current AP an FT-OWE Request. The request is an action frame, and includes the STA MAC address, the target AP BSSID, the RSNE with the PMKROName, the MDE, and the FTE with a SNonce and the ROKH-ID. The current AP passes the request, over the DS, to the target AP. The target AP responds with a FT Response containing the STA MAC address, the target AP BSSID, the RSNE with the PMKROName, the MDE, and the FTE with a MIC (as defined in [RFC8110] section 4.4), ANonce, SNonce, R1KH-ID for the target AP, and ROKH-ID. The current AP relays this response to the STA through an FT Response action frame. The STA responds with an FT confirm action frame sent to the current AP. The

frame includes the STA MAC address, the target AP BSSID, the RSNE with the PMKR1Name (derived from the PMKR0Name, the R1KH-ID obtained above from the target AP, and the STA MAC address (S1KH-ID), as defined in [IEEE802.11] clause 12.7.1.4.1), the MDE, and the FTE with a MIC (as defined in [RFC8110] section 4.4), ANonce, SNonce, R1KH-ID for the target AP, and the R0KH-ID. The current AP forwards this message over the Distribution System to the target AP. The target AP then replies with an FT ACK message, that includes the STA MAC address, the target AP BSSID, the RSNE with the PMKR1Name, the MDE, the FTE with a MIC ((as defined in [RFC8110] section 4.4), the ANonce, Snonce, R1KH-ID and R0KH-ID, and the Timeout Interval Element (TIE) that specifies the reassociation deadline value. At this stage, the FT-OWE authentication to the target AP has completed, and stays valid until the expiration of the reassociation deadline time. It is understood that the STA may establish such authentication to multiple target APs. Later, when the STA needs to associate to the target AP and proceed to secure exchanges, and while the authentication is still valid, the STA proceeds through the FT-OWE reassociation exchange with the target AP as described in section 4.4.1.

5. IANA Considerations

This document does not require any IANA actions.

6. Security Considerations

FT-OWE does not provide authentication. FT-OWE provides a cryptographic assurance that a target AP with which a STA has established an FT-OWE connection is derived from an FT-OWE connection to its current AP, assuring each AP (current and target) have a trusted network connection with each other, and thus the information provided by both APs are likely coherent. FT-OWE does not guarantee that the APs are legitimate. When a large set of APs provide coherent information and allow for FT-OWE communication, it is likely that all these APs are part of the same system. The system may be legitimate or not. OWE is not a replacement for any authentication protocol specified in [IEEE802.11] and is not intended to be used when an alternative that provides real authentication is available.

Authors' Addresses

Jerome Henry
Cisco

Email: jerhenry@cisco.com

Stephen Orr
Cisco

Email: sorr@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 27 June 2022

B. Jordan, Ed.
Broadcom
S. Erdtman
Spotify AB
A. Rundgren
Independent
24 December 2021

JWS Clear Text JSON Signature Option (JWS/CT)
draft-jordan-jws-ct-07

Abstract

This document describes a method for extending the scope of the JSON Web Signature (JWS) specification, called JWS/CT (JWS "Clear Text"). By combining the detached mode of JWS with the JSON Canonicalization Scheme (JCS), JWS/CT enables JSON objects to remain in the JSON format after being signed. In addition to supporting a consistent data format, this arrangement also simplifies documentation, debugging, and logging. The ability to embed signed JSON objects in other JSON objects, makes the use of counter-signatures straightforward.

This informational specification has been produced outside the IETF, is not an IETF standard, and does not have IETF consensus. The intended audiences of this document are JSON tool vendors as well as designers of JSON-based cryptographic solutions.

Status of This Memo

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

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

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

This Internet-Draft will expire on 27 June 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Detailed Operation	4
3.1. Signature Creation	4
3.1.1. Create the JSON Object to be Signed	4
3.1.2. Canonicalize the JSON Object to be Signed	5
3.1.3. Generate a JWS String	5
3.1.4. Assemble the Signed JSON Object	5
3.2. Signature Validation	6
3.2.1. Parse the Signed JSON Object	6
3.2.2. Fetch the Signature Property String	6
3.2.3. Remove the Signature Property String	6
3.2.4. Canonicalize the Remaining JSON Object	7
3.2.5. Validate the JWS String	7
4. IANA Considerations	8
5. Security Considerations	8
6. References	9
6.1. Normative References	9
6.2. Informative References	9
Appendix A. Open-Source Implementations	10
Appendix B. JWS/CT Application Notes	10
B.1. Counter-Signatures	10
B.2. Detached Signatures	12
B.3. Array of Signatures	13
Appendix C. Test Vector Using the ES256 Algorithm	14
Appendix D. Enhanced JWS Processing Option	15
Acknowledgements	15
Document History	15
Authors' Addresses	16

1. Introduction

This specification introduces a method for augmenting data expressed in the JSON [RFC8259] notation, with enveloped signatures, similar to the scheme used in XML Signature [XMLDSIG]. For interoperability reasons this specification constrains JSON objects to the I-JSON [RFC7493] subset.

To avoid "reinventing the wheel", this specification leverages JSON Web Signature (JWS) [RFC7515].

By building on the detached mode of JWS in combination with the JSON Canonicalization Scheme (JCS) [RFC8785], JSON objects to be signed can be kept in the JSON format. This arrangement is here referred to as JWS/CT, where CT stands for "Clear Text" signing.

The primary motivations for keeping signed JSON objects in the JSON format include simplified documentation, debugging, and logging, as well as for maintaining a consistent message structure.

Another target is HTTP-based signature schemes that currently utilize HTTP header values for holding detached signatures. By using the method described herein, signed JSON-formatted HTTP requests and responses may be self-contained and thus be serializable. The latter facilitates such data to be

- * stored in databases
- * passed through intermediaries
- * embedded in other JSON objects
- * counter-signed

without losing the ability to (at any time) verify signatures.

Appendix B outlines different ways to handle multiple signatures including counter-signing using JWS/CT.

The intended audiences of this document are JSON tool vendors as well as designers of JSON-based cryptographic solutions.

2. Terminology

Note that this document is not on the IETF standards track. However, a conformant implementation is supposed to adhere to the specified behavior for security and interoperability reasons. This text uses BCP 14 to describe that necessary behavior.

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

3. Detailed Operation

This section describes the details related to signing and validating signatures based on this specification.

The following characteristics are crucial to know for prospective JWS/CT implementers and users:

- * With the exception of the reliance on the detached mode described in Appendix F of JWS [RFC7515], JWS/CT does not alter the JWS signature creation process, validation process, or format. This means that the contents of JWS headers as well as things related to signature algorithms and cryptographic keys are out of scope for this specification. A slightly enhanced processing option is outlined in Appendix D.
- * JWS/CT depends exclusively on the JWS Compact Serialization mode.
- * JSON data to be signed MUST be supplied as JSON objects. That is, direct signing of JSON arrays or JSON primitives is out of scope for this specification.
- * JCS [RFC8785] constrains JSON objects to the I-JSON [RFC7493] subset.

The signature creation and signature validation sections (Section 3.1 and Section 3.2 respectively), feature examples using the HS256 JOSE algorithm [RFC7518] with a 256-bit key having the following value, here expressed as hexadecimal bytes:

```
7f dd 85 1a 3b 9d 2d af c5 f0 d0 00 30 e2 2b 93
43 90 0c d4 2e de 49 48 56 8a 4a 2e e6 55 29 1a
```

3.1. Signature Creation

The following sub-sections describe how JSON objects can be signed according to the JWS/CT specification.

3.1.1. Create the JSON Object to be Signed

Create or parse the JSON object to be signed.

The following example object is used to illustrate the operations in the sections that follow:

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2000, true]
}
```

3.1.2. Canonicalize the JSON Object to be Signed

Use the result of the previous step as input to the canonicalization process described in JCS [RFC8785].

Applied to the example, the following JSON string should be generated:

```
{"otherProperties":[2000,true],"statement":"Hello signed world!"}
```

After encoding the string above in the UTF-8 [UNICODE] format, the following bytes (here in hexadecimal notation) should be generated:

```
7b 22 6f 74 68 65 72 50 72 6f 70 65 72 74 69 65 73 22 3a 5b 32 30
30 30 2c 74 72 75 65 5d 2c 22 73 74 61 74 65 6d 65 6e 74 22 3a 22
48 65 6c 6c 6f 20 73 69 67 6e 65 64 20 77 6f 72 6c 64 21 22 7d
```

3.1.3. Generate a JWS String

Use the result of the previous step as JWS Payload to the signature process described in Appendix F of JWS [RFC7515].

For the example, the JWS header is assumed to be:

```
{"alg":"HS256"}
```

The resulting JWS string should then after payload removal and using the key specified in Section 3, read as follows:

```
eyJhbGciOiJIUzI1NiJ9..VHVItCBcb8Q5CI-49imarDtJeSxH2uLU0DhqQP5Zjw4
```

3.1.4. Assemble the Signed JSON Object

Before a complete signed object can be created, a dedicated top-level property for holding the JWS signature string needs to be defined. The only requirement is that this property MUST NOT clash with any other top-level property name. The JWS string itself MUST be supplied as a JSON string argument to the signature property.

For the example, the property name "signature" is assumed to be the designated holder of the JWS string. Equipped with a signature property, the JWS string from the previous section, and the original JSON example, the process above should result in the following, now signed JSON object (with a line break in the "signature" property for display purposes only):

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2000, true],
  "signature": "eyJhbGciOiJIUzI1NiJ9..VHVItCBCb8Q5CI-49imar
DtJeSxH2uLU0DhqQP5Zjw4"
}
```

3.2. Signature Validation

The following sub-sections describe how JSON objects signed according to the JWS/CT specification can be validated.

3.2.1. Parse the Signed JSON Object

Parse the JSON object that is expected to have been signed. If the parsing is unsuccessful, the operation MUST cause a compliant implementation to terminate processing and return an error indication.

To illustrate the subsequent operations the signed JSON object featured in Section 3.1.4 is used as example.

3.2.2. Fetch the Signature Property String

After successful parsing, retrieve the designated JSON top-level property holding the JWS string. If the property is missing or its argument is not a JSON string value, the operation MUST cause a compliant implementation to terminate processing and return an error indication.

For the example, where the property named "signature" is assumed to hold the JWS string, the operation above should return the following string:

```
eyJhbGciOiJIUzI1NiJ9..VHVItCBCb8Q5CI-49imarDtJeSxH2uLU0DhqQP5Zjw4
```

3.2.3. Remove the Signature Property String

Since the signature is calculated over the actual JSON object data, the designated signature property and its argument MUST be removed from the signed JSON object.

If applied to the example the resulting JSON object should read as follows:

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2000, true]
}
```

Note: JSON tools usually by default remove whitespace. In addition, the original ordering of properties may not always be honored. However, none of this has (due to the canonicalization performed by JCS), any impact on the result.

3.2.4. Canonicalize the Remaining JSON Object

Use the result of the previous step as input to the canonicalization process described in JCS [RFC8785].

If applied to the example the result of the process above should read as follows:

```
{"otherProperties":[2000,true],"statement":"Hello signed world!"}
```

After encoding the string above in the UTF-8 [UNICODE] format, the following bytes (here in hexadecimal notation) should be generated:

```
7b 22 6f 74 68 65 72 50 72 6f 70 65 72 74 69 65 73 22 3a 5b 32 30
30 30 2c 74 72 75 65 5d 2c 22 73 74 61 74 65 6d 65 6e 74 22 3a 22
48 65 6c 6c 6f 20 73 69 67 6e 65 64 20 77 6f 72 6c 64 21 22 7d
```

3.2.5. Validate the JWS String

After extracting the detached mode JWS string and canonicalizing the JSON object (to retrieve the JWS Payload), the JWS string MUST be restored as described in Appendix F of JWS [RFC7515]. The actual JWS validation procedure is not specified here because it is covered by [RFC7515] and also depends on application-specific policies like:

- * Accepted JWS signature algorithms
- * Accepted and/or required JWS header elements
- * Signature key lookup methods

If the validation process for some reason fails, the operation MUST cause a compliant implementation to terminate processing and return an error indication.

For the example, validation is straightforward since both the algorithm and the key to use are predefined (see Section 3). The input string to a JWS validator should after the process step above read as follows (with line breaks for display purposes only):

```
eyJhbGciOiJIUzI1NiJ9.eyJvdGhlclByb3BlcnRpZXMlOlsyMDAwLHRydWVdLCJzdGF0ZWllbnQiOiJIZWxsbyBzaWduZWQgd29ybGQhIn0.VHVItCBCb8Q5CI-49imarDtJeSxH2uLU0DhqQP5Zjw4
```

4. IANA Considerations

This document has no IANA actions.

5. Security Considerations

This specification inherits all the security considerations of JWS [RFC7515] and JCS [RFC8785].

In similarity to any other signature specification, it is crucial that signatures are verified before acting on the signed payload.

However, poorly tested software components may also introduce security issues. Consider the following JSON example:

```
{
  "fromAccount": "1234",
  "toAccount": "4567",
  "amount": {
    "value": 100,
    "currency": "USD"
  }
}
```

A non-compliant JCS implementation could return

```
{"amount": {}, "fromAccount": "1234", "toAccount": "4567"}
```

giving an attacker the ability to change "amount" to whatever it wants. Note though that this attack presumes that the consumer and producer use implementations broken in the same way, otherwise the signature would not validate.

For usage in a wider community, the name of the designated signature property becomes a critical factor that **MUST** be documented and communicated. However, in a properly designed system, a faulty or missing signature **MUST** "only" lead to failed operation, and not to a security breach.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<https://www.rfc-editor.org/info/rfc7518>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <<https://www.rfc-editor.org/info/rfc8785>>.
- [UNICODE] The Unicode Consortium, "The Unicode Standard", <<https://www.unicode.org/versions/latest/>>.

6.2. Informative References

- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.

- [RFC7797] Jones, M., "JSON Web Signature (JWS) Unencoded Payload Option", RFC 7797, DOI 10.17487/RFC7797, February 2016, <<https://www.rfc-editor.org/info/rfc7797>>.
- [SHS] NIST, "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015, <<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>>.
- [XMLDSIG] W3C, "XML Signature Syntax and Processing Version 1.1", W3C Recommendation, April 2013, <<https://www.w3.org/TR/xmlsig-core1/>>.

Appendix A. Open-Source Implementations

Due to the simplicity of this specification, there is hardly a need for specific support software. However, JCS which is (at the time of writing), a relatively new design, may be fetched as a separate component for multiple platforms. The following open-source implementations have been verified to be compatible with JCS:

- * JavaScript: <<https://www.npmjs.com/package/canonicalize>>
- * Java: <<https://mvnrepository.com/artifact/io.github.erdtman/java-json-canonicalization>>
- * Go: <<https://github.com/cyberphone/json-canonicalization/tree/master/go>>
- * .NET/C#: <<https://github.com/cyberphone/json-canonicalization/tree/master/dotnet>>
- * Python: <<https://github.com/cyberphone/json-canonicalization/tree/master/python3>>

Appendix B. JWS/CT Application Notes

The following application notes are not a part of the JWS/CT core; they show how JWS/CT can be used in contexts involving multiple signatures.

B.1. Counter-Signatures

Consider the following JWS/CT object showing an imaginary real estate business record (with a line break in the "signature" property for display purposes only):

```
{
  "gps": [38.89768255588178, -77.03658644893932],
  "object": {
    "type": "house",
    "price": "$635,000"
  },
  "role": "buyer",
  "name": "John Smith",
  "timeStamp": "2020-11-08T13:56:08Z",
  "signature": "eyJhbGciOiJIUzI1NiJ9..z1PMniQiz4Eie86oK4xo25z
uyW92csiDqyiQrF6R5ug"
}
```

The signature above was created using the example key from Section 3.

Adding a notary signature on top of this could be performed by embedding the former object as follows (with line breaks in the "signature" properties for display purposes only):

```
{
  "attesting": {
    "gps": [38.89768255588178, -77.03658644893932],
    "object": {
      "type": "house",
      "price": "$635,000"
    },
    "role": "buyer",
    "name": "John Smith",
    "timeStamp": "2020-11-08T13:56:08Z",
    "signature": "eyJhbGciOiJIUzI1NiJ9..z1PMniQiz4Eie86oK4xo25z
uyW92csiDqyiQrF6R5ug"
  },
  "role": "notary",
  "name": "Carol Lombardi-Jones",
  "timeStamp": "2020-11-08T13:58:42Z",
  "signature": "eyJhbGciOiJFUzI1NiJ9..AVmJGUWp1JD0pf2jl_UQWXbf-
qj-2RWxOnyAXihd4POKbnjWqqSBmHPNfgMQFH_s5sXHkIOkDZe2nShqEJOEVA"
}
```

A side effect of this arrangement is that the notary's signature signs not only the notary data, but the buyer's data and signature as well. In most cases this way of adding signatures is advantageous since it maintains the actual order of signing events which also cannot be tampered with without invalidating the outermost signature.

Note that all properties above including "signature" are application specific.

The notary's signature was created using the example key from Appendix C.

B.2. Detached Signatures

In the case the signing entities are "peers" or are unrelated to each other, counter-signatures like described in Appendix B.1 are not applicable since they presume a specific flow. For supporting independent or asynchronous signers targeting a common document or data object, an imaginable solution is using a scheme where each signer calculates a hash of the target document/data and includes the hash together with signer-specific meta data like the following:

```
{
  <<Common Document/Data to Sign...>>

  "signers": [{
    "sha256": "<<Hash of Document/Data to Sign>>",

    <<Signer-related meta data...>>

    "signature": "<<Signer JWS Signature>>"
  }, {
    "sha256": "<<Hash of Document/Data to Sign>>",

    <<Signer-related meta data...>>

    "signature": "<<Signer JWS Signature>>"
  }]
}
```

In this case the object to sign would not be limited to JSON; it could, for example, be a PDF document hosted on a specific URL. Note that the relying party would have to update the structure for each signature received. In some cases a database would probably be more useful for holding individual signatures since a database can cope with any number of signers as well as keeping track of who have actually signed. The latter is crucial for things like international treaties and company board statements.

Note that although "signers", "sha256", and "signature" are application specific property names, the objects in the "signers" array are assumed to be fully conformant with the JWS/CT specification.

The following example shows a possible detached signature solution (with line breaks in the "signature" properties for display purposes only):

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2000, true],
  "signers": [{
    "sha256": "n-i0HIBJKELoTicCK9c5nqJ8cYH0znGRcEbYKoQfm70",
    "timeStamp": "2020-11-18T07:45:28Z",
    "name": "Alice",
    "signature": "eyJhbGciOiJIUzI1NiJ9..AE7CnzSYsasPE3yrdsAwi
avd3IdWtdAmDE8FRMwYLA8"
  }, {
    "sha256": "n-i0HIBJKELoTicCK9c5nqJ8cYH0znGRcEbYKoQfm70",
    "timeStamp": "2020-11-18T08:03:40Z",
    "name": "Bob",
    "signature": "eyJhbGciOiJIUzI1NiJ9..0tNLy0pLcHUjPhhorpKd5
7a8zTPeqlrOjATiSlPQlvcIE99x6mHmow04tPbJS8dqSqO9c4RkKW6jeL4ZyWpXLA"
  }]
}
```

Notes:

- * "Alice" used the example key from Section 3 while "Bob" used the example key specified in Appendix C.
- * The "sha256" properties hold base64url-encoded [RFC4648], SHA256-hashes [SHS] of the canonicalized data created in Section 3.1.2.
- * This arrangement requires a two-step validation process where each JWS/CT object in the "signers" array is individually validated, as well as having its "sha256" property compared with the actual hash of the canonicalized common data.

B.3. Array of Signatures

Another possibility supporting multiple and independent signatures is collecting JWS signature strings in a JSON array object according to the following scheme:

```
{
  <<Common Document/Data to Sign...>>

  "<<Signature property>>": [ "<<Signature-1>>",
                                "<<Signature-2>>",
                                .
                                "<<Signature-n>>" ]
}
```

Processing would follow Section 3, with the addition that each signature is dealt with individually.

Compared to Appendix B.2, signature arrays imply that possible signer-specific meta-data is supplied as JWS extensions in the associated signature's base64url-encoded header.

By combining the example used in Section 3 with the test vector in Appendix C, a valid signature array object could be as follows (with line breaks in the "signatures" property for display purposes only):

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2000, true],
  "signatures": ["eyJhbGciOiJIUzI1NiJ9..VHVItCBCb8Q5CI-49imar
DtJeSxH2uLU0DhqQP5Zjw4",
                 "eyJhbGciOiJFUzI1NiJ9..ENP0j0-QPsA7N_Mg1-RMN
9IxapeTWtQwR7sPUqEiSNHPuV_fqSdRqqkLOlBdV0lcc4lSJdn1XCv-ZHYdZ9t3kA"]
}
```

Note that "signatures" is not a keyword, it was only selected to highlight the fact that there are multiple signatures.

Appendix C. Test Vector Using the ES256 Algorithm

This appendix shows how a signed version of the JSON example object in Section 3.1.1 would look like if applying the ES256 JOSE algorithm [RFC7518] (with a line break in the "signature" property for display purposes only):

```
{
  "statement": "Hello signed world!",
  "otherProperties": [2000, true],
  "signature": "eyJhbGciOiJFUzI1NiJ9..ENP0j0-QPsA7N_Mg1-RMN
9IxapeTWtQwR7sPUqEiSNHPuV_fqSdRqqkLOlBdV0lcc4lSJdn1XCv-ZHYdZ9t3kA"
}
```

The example above depends on a JWS header holding the algorithm {"alg":"ES256"}, and the following private key, here expressed in the JWK [RFC7517] format:

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "6BKxpty8cI-exDzCkh-goU6dXq3MbcY0cd1LaAxiNrU",
  "y": "mChcvUzm44j3Lt2b5BPyQloQ91tf2D2V-gzeUxWaUdg",
  "d": "6XxMFXhcYT5QN9w5TIg2aSKsbcj-pj4BnZkK7Zot4B8"
}
```

Note that signing with the ES256 algorithm returns different results for each signature due to a randomization step in the signature computation process.

Appendix D. Enhanced JWS Processing Option

By default, JWS/CT uses the JWS compact serialization mode "as is". As a consequence, a technically redundant, internal-only, base64url encoding step is performed over the JWS Payload. Although the performance hit should be marginal for most real-world applications, a possibility is using the "Unencoded Payload" mode of RFC7797 [RFC7797]. However, this requires that the JWS implementation supports the "b64":false and "crit":["b64"] header elements implied by RFC7797, effectively rendering the RFC7797 mode as an implementer option for specific communities.

Acknowledgements

People who have contributed directly and indirectly with valuable input to this specification include Vladimir Dzhuvinov, Freddi Gyara, and Filip Skokan.

Document History

[[This section to be removed by the RFC Editor before publication as an RFC]]

Version 00:

- * Initial publication.

Version 01:

- * Added paragraph to Abstract.

- * Updated Security Considerations.

Version 02:

- * Changed alternative test key to ES256/P-256.
- * Moved RFC7797 to an appendix.
- * Changed <tt> to only be used on keywords.
- * Added some clarity to detached signatures.

Version 03:

- * Language changes suggested by ISE.

Version 04:

- * Language nit.

Version 05:

- * Document refresh.

Version 06:

- * Changes after ISE review.

Version 07:

- * Changes after ISE and external reviews.

Authors' Addresses

Bret Jordan (editor)
Broadcom
1320 Ridder Park Drive
San Jose, CA 95131
United States of America

Email: bret.jordan@broadcom.com

Samuel Erdtman
Spotify AB
Birger Jarlsgatan 61, 4tr
SE-113 56 Stockholm
Sweden

Email: erdtman@spotify.com

Anders Rundgren
Independent
Montpellier
France

Email: anders.rundgren.net@gmail.com

URI: <https://www.linkedin.com/in/andersrundgren/>

mls
Internet-Draft
Intended status: Informational
Expires: 8 September 2022

M. Knodel
CDT
F. Baker

O. Kolkman
ISOC
S. Celi
Cloudflare
G. Grover

Centre for Internet and Society
7 March 2022

Definition of End-to-end Encryption
draft-knodel-e2ee-definition-03

Abstract

End-to-end encryption (E2EE) is an application of cryptography in communications systems between endpoints. E2EE systems are unique in providing features of confidentiality, integrity and authenticity for users. Improvements to E2EE strive to maximise the system's security while balancing usability and availability. Users of E2EE communications expect trustworthy providers of secure implementations to respect and protect their right to whisper.

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Formal definition of end-to-end encryption	3
2.1. End point	3
2.2. End-to-end principle	4
2.3. Encryption	6
2.4. Succinct definition of end-to-end security	7
3. End-to-end encrypted systems design	7
3.1. Features	7
3.1.1. Necessary features	8
3.1.2. Optional/desirable features	8
3.2. Challenges	9
4. End-user expectations	10
4.1. A conversation is confidential	11
4.2. Providers are trustworthy	11
4.3. Access by a third-party is impossible	11
4.4. Pattern inference is minimised	12
4.5. The E2EE system is not compromised	12
5. Conclusions	12
6. Acknowledgements	13
7. Security Considerations	13
8. IANA Considerations	13
9. Informative References	13
Authors' Addresses	14

1. Introduction

This document defines end-to-end encryption (E2EE) using three different dimensions that together comprise a full definition of E2EE, which can be applied in a variety of contexts.

The first is a formal definition that draws on the basic understanding of end points and cryptography. The second looks at E2EE systems from a design perspective, both its fundamental features and the direction of travel towards improving those features. Lastly we consider the expectations of the user of E2EE systems.

These dimensions taken as a whole comprise a generally comprehensible picture of consensus at the IETF as to what is end-to-end encryption, irrespective of application, from messaging to video conferencing, and between any number of end points.

2. Formal definition of end-to-end encryption

An end-to-end encrypted communications system, irrespective of the content or the specific methods employed, relies on two important and rigorous technical concepts: The end-to-end principle and what defines an end, according to the IETF because of its importance to internet protocols; and encryption, an application of cryptography and the primary means employed by the IETF to secure internet protocols. In the tradition of cryptography it's also possible to achieve a succinct definition of end-to-end encrypted security.

2.1. End point

Intuitively, an "end" either sends messages or receives them, usually both; other systems on the path are just that - other systems.

It is, however, not trivial to establish the definition of an end point in isolation, because its existence inherently depends on at least one other entity in a communications system. That is why we will now move directly into an analysis of the end-to-end principle, which introduces nuance, described in the following sub-section.

However despite the nuance for engineers, it is now widely accepted that the communication system itself begins and ends with the user [RFC8890]. We imagine people (through an application's user interface, or user agent) as components in a subsystem's design. An important exception to this in E2EE systems might be the use of public key infrastructure where a third party is often used in the authentication phase to enhance the larger system's trust model. Responsible use of public key infrastructure is required in such cases, such that the E2EE system does not admit third parties under the user's identity.

We cannot equate user agent and user, yet we also cannot fully separate them. As user-agent computing becomes more complex and often more proprietary, the user agent becomes less of an "advocate" for the best interests of the user. This is why we focus in a later section on the E2EE system being able to fulfill user expectations.

2.2. End-to-end principle

We need first to answer "What constitutes an end?", which is an important question in any review of the End-to-End Principle [RFC3724]. However the notion of an end point is more fully defined within the principle of end-to-end communications.

In 1984 the "end-to-end argument" was introduced [saltzer] as a design principle that helps guide placement of functions among the modules of a distributed computer system. It suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level. It is used to design around questions about which parts of the system should make which decisions, and as such the identity of the actual "speaker" or "end" may be less obvious than it appears. The communication described by Saltzer is between communicating processes, which may or may not be on the same physical machine, and may be implemented in various ways. For example, a BGP speaker is often implemented as a process that manages the Routing Information Base (RIB) and communicates with other BGP speakers using an operating system service that implements TCP. The RIB manager might find itself searching the RIB for prefixes that should be advertised to a peer, and performing "writes" to TCP for each one. TCP in this context often implements a variant of the algorithm described in RFC 868 (the "Nagle algorithm"), which accumulates writes in a buffer until there is no data in flight between the communicants, and then sends it - which might happen several times during a single search by the RIB manager. In that sense, the RIB manager might be thought of as the "end", because it decides what should be communicated, or TCP might be the "end", because it actually sends the TCP Segment, detects errors if they occur, retransmits it if necessary, and ultimately decides that the segment has been successfully transferred.

Another important question is "what statement exactly summarizes the end-to-end principle?". Saltzer answered this in two ways, the first of which is that the service implementing the transaction is most correct if it implements the intent of the application that sent it, which would be to move the message toward the destination address in the relevant IP header. Salzer's more thorough treatment, however, deals with end cases that come up in implementation: "Examples discussed in the paper", according to the abstract, "include bit error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgement." It also notes that there is occasionally a rationale for ignoring the end-to-end arguments for the purposes of optimization. There may be other user expectations or design features, some explained below, which need to be balanced with the end-to-end argument.

More concisely, suppose that an end user is the end identity. An E2EE system may run between potential end points at different network layers within the end identity's possession. These end points may then be considered acceptable sub-identities provided that no path between the end identity and sub-identity is accessible by any third party. There are quite a number of examples of common situations where tunnels are used and this does not apply. For instance, the examples below all provide encryption by which data is turned into clear text in locations that are not under control of the end user:

- * The common VPNS business model whereby a TLS or an IPsec tunnel terminates at the service provider's server and is subsequently forwarded to its destination elsewhere in unencrypted form;
- * Email transport whereby an unencrypted message is traverses from sending mail user agent, between various mail transfer agents, and finally to the a receiving mail user agent, all over TLS protected connections;
- * The encrypted connection of last mile connections such as those in 4G LTE;

This definition of end points accounts for potentially several devices owned by a user, and various application-specific forwarding or delivery options among them. It also accounts for E2EE systems running at different network layers. Regardless of the sub-identities allowed, the definition is contingent on that all end sub-identities are under the end identity's control and no third party (or their sub-identities, e.g. system components under third-party control) can access the end sub-identities nor links between the sub-identity and end identity. This creates a tree hierarchy with the end user as the root at the top, and all potential end points being

under their direct control, without third party access. As an example, decryption at organizational network router before message forwarding (encrypted or unencrypted) to the end identity does not constitute E2EE. However, E2EE to a user's personal device and subsequent E2EE message forwarding to another one of the user's personal devices (without access available to any third party at any link or on device) maintains E2EE data possession for the user.

2.3. Encryption

From draft-dkg-hrpg-glossary-00, encryption is fundamental to the end-to-end principle. "End-to-End : The principal of extending characteristics of a protocol or system as far as possible within the system. For example, end-to-end instant message encryption would conceal communication content from one user's instant messaging application through any intermediate devices and servers all the way to the recipient's instant messaging application. If the message was decrypted at any intermediate point-for example at a service provider-then the property of end-to-end encryption would not be present." [dkg] Note that this only talks about the contents of the communication and not the metadata generated from it.

The way to achieve a truly end-to-end communications system is indeed to encrypt the content of the data exchanged between the endpoints, e.g. sender(s) and receiver(s). The more common end-to-end technique for encrypting uses a double-ratchet algorithm with an authenticated encryption scheme, present in many modern messenger applications such as those considered in the IETF Messaging Layer Security working group, whose charter is to create a document that satisfies the need for several Internet applications for group key establishment and message protection protocols [mls]. OpenPGP, mostly used for email, uses a different technique to achieve encryption. It is also chartered in the IETF to create a specification that covers object encryption, object signing, and identity certification [openpgp]. Both protocols rely on the use of asymmetric and symmetric encryption, and exchange public keys with amongst end points.

There are dozens of documents in the RFC Series that fundamentally and technically define encryption schemes. Perhaps interesting work to be done would be to survey all existing documents of this kind to define, in aggregate, their common features. The point is, the IETF has clear mandate and demonstrated expertise in defining the specifics of encrypted communications of the internet.

While encryption is fundamental to the end-to-end principle, it does not stand alone. As in the history of all security, authentication and data integrity properties are also linked, and contributed to the end-to-end nature of E2EE. Permission of data manipulation or

pseudo-identities for third parties to allow access under the user's identity are against the intention of E2EE. Thus, end point authenticity must be established as (sub-)identities of the end user, and end-to-end integrity must also be maintained by the system. There is considerable system design flexibility available in entity authentication mechanisms and data authentication that still meet this requirement.

2.4. Succinct definition of end-to-end security

A succinct definition for end-to-end security can describe the security of the system by the probability of an adversary's success in breaking the system. Example snippet:

The adversary successfully subverts an end-to-end encrypted system if it can succeed in either of the following: 1) the adversary can produce the participant's local state (meaning the adversary has learned the contents of participant's messages), or 2) the states of conversation participants do not match (meaning that the adversary has influenced their communication in some way). To prevent the adversary from trivially winning, we do not allow the adversary to compromise the participants' local state.

We can say that a system is end-to-end secure if the adversary has negligible probability of success in either of these two scenarios [komlo].

3. End-to-end encrypted systems design

When looking at E2EE systems from a design perspective, the first consideration is the list of fundamental features that distinguish an E2EE system from one that does not employ E2EE. Secondly one must consider the direction of travel for improving the features of E2EE systems. In other words, what challenges are the designers, developers and implementers of E2EE systems facing?

The features and challenges listed below are framed holistically rather than from the perspective of their design, development, implementation or use.

3.1. Features

Defining a technology can also be done by inspecting what it does, or is meant to do, in the form of features. The features of end-to-end encryption from an implementation perspective can be inspected across several important categories: 1) the necessary features of E2EE of authenticity, confidentiality, and integrity, whereas features of 2) availability, deniability, forward secrecy, and post-compromise

security are enhancements to E2EE systems.

3.1.1. Necessary features

Authenticity A system provides message authenticity if the recipient and sender agree on each other's identities and the contents of their communications.

Confidentiality A system provides message confidentiality if only the sender and intended recipient(s) can read the message plaintext, i.e. messages are encrypted by the sender such that only the intended recipient(s) can decrypt them.

Integrity A system provides message integrity when it guarantees that messages that have been modified in transit can be detected reliably, i.e. a recipient is assured that a message cannot be undetectably modified in any way.

3.1.2. Optional/desirable features

Availability A system provides high availability if the user is able to get to the message when they so desire and potentially from more than one device, i.e. a message arrives to a recipient even if they have been offline for a long time.

Deniability Deniability ensures that anyone with a record of the transcript, including message recipients, cannot cryptographically prove to others that a particular participant of a communication authored the message. As demonstrated by the Signal and OTR protocols, this optional property must exist in conjunction with the necessary property of message authenticity, i.e. participants in a communication must be assured that they are communicating with the intended parties but this assurance cannot be proof to any other parties.

Forward secrecy Forward secrecy is a security property that prevents attackers from decrypting encrypted data they have previously captured over a communication channel before the time of compromise, even if they have compromised one of the endpoints. Forward secrecy is usually achieved by updating the encryption/decryption keys, and older ones are deleted periodically.

Post-compromise security Post-compromise security is a security

property that seeks to guarantee a way to recover from an end-point compromise (and consequently that communication sent post-compromise is protected with the same security properties that existed before the compromise). It is usually achieved by adding ephemeral key exchanges to the derivation of encryption/decryption keys.

Metadata obfuscation Steps should be taken to minimize metadata such as user obfuscating IP addresses, reducing non-routing metadata, and avoiding extraneous message headers can enhance the confidentiality and security features of E2EE systems.

3.2. Challenges

Earlier we defined end-to-end encryption using formal definitions assumed by internet protocol implementations. Also because "the IETF is a place for state-of-the-art producing high quality, relevant technical documents that influence the way people design, use, and manage the Internet" we can be confident that current deployments of end-to-end encrypted technologies in the IETF indicate the cutting edge of their developments, yet another way to define what is, or ideally should be, how a technology is defined.

Below is an exhaustive, yet vaguely summarised, list of the challenges currently faced by protocol designers of end-to-end encrypted systems. In other words, in order to realise the goals of end-to-end encrypted systems, both for users and implementers (see previous section), these problems must be tackled. Problems that fall outside of this list are likely 1) unnecessary feature requests that negligibly, or do nothing to, achieve the aims of end-to-end encrypted systems or are 2) in some way antithetical to the goals of end-to-end encrypted systems.

Public key verification is very difficult for users to manage. Authentication of the two ends is required for confidential conversations. Therefore solving the problem of verification of public keys is a major concern for any end-to-end encrypted system design. Some applications bind together the account identity and the key, and leave users to establish a trust relationship between them, assisted by public key fingerprint information.

Users want to smoothly switch application use between devices, but this comes at a cost to the security of user data. Thus, there is a problem of availability in end-to-end encrypted systems because the account identity's private key is generated by and stored on the end-user's original device and to move the private key to another device compromises the security of one of the end-points of the system.

Existing protocols are vulnerable to meta-data analysis, even though meta-data is often much more sensitive than content. Meta-data is plaintext information that travels across the wire and includes delivery-relevant details that central servers need such as the account identity of end-points, timestamps, message size. Meta-data is difficult to obfuscate efficiently.

Users need to communicate in groups, but this presents major problems of scale for end-to-end encryption systems that rely on public key cryptography.

The whole of a user's data should remain secure if only one message is compromised. However, for encrypted communication, you must currently choose between forward secrecy or the ability to communicate asynchronously. This presents a problem for application design that uses end-to-end encryption for asynchronous messaging over email, RCS, etc.

Users of E2EE systems should be able to communicate with any medium of their choice, from text to large files, however there is often a resource problem because there are no open protocols to allow users to securely share the same resource in an end-to-end encrypted system. Client-side, e.g. end-point, activities like URL unfurling scanning.

Usability considerations are sometimes in conflict with security considerations, such as message read status, typing indicators, URL/link previews.

Deployment is notoriously challenging for any software application where maintenance and updates can be particularly disastrous for obsolete cryptographic libraries.

4. End-user expectations

While the formal definition and properties of an E2EE system relate to communication security, they do not draw from a comprehensive threat model or speak to what users expect from E2EE communication. It is in this context that some E2EE designs and architectures may ultimately run contrary to user expectations of E2EE systems [GEC-EU]. Although some system designs do not directly violate "the math" of encryption algorithms, they do so by implicating and weakening other important aspects of an E2EE _system_.

4.1. A conversation is confidential

Users talking to one another in an E2EE system should be the only ones that know what they are talking about [RFC7624]. People have the right to data privacy as defined in international human rights law and within the right to free expression and to hold opinions is inferred the right to whisper, whether or not they are using digital communications or walking through a field.

4.2. Providers are trustworthy

While "trustworthy" can be rigourously defined from an engineering perspective, for the purposes of this document we choose a definition of Trustworthy inspired by an internal workshop by Internet Society staff:

Trustworthy A system is completely trustworthy if and only if it is completely resilient, reliable, accountable, and secure in a way that consistently meets users' expectations. The opposite of trustworthy is untrustworthy.

This definition is complete in its positive and negative aspects: what it is, e.g. "Worthy of confidence" and what it is not, e.g. in RFC 7258: "behavior that subverts the intent of communicating parties without the agreement of those parties" [RFC7258].

Therefore, a trustworthy end-to-end encrypted communication system is the set of functions needed by two or more parties to communicate among each other in a confidential and authenticated fashion without any third party having access to the content of that communication where the functions that offer the confidentiality and authenticity are trustworthy.

4.3. Access by a third-party is impossible

No matter the specifics, any methods used to access to the content of the messages by a third party would violate a user's expectations of E2EE messaging. "[T]hese access methods scan message contents on the user's [device]", which are then "scanned for matches against a database of prohibited content before, and sometimes after, the message is sent to the recipient" [GEC-EU]. Third party access also covers cases without scanning - namely, it should not be possible for any third-party end point to access the data regardless of reason.

If a method makes private communication, intended to be sent over an encrypted channel between end points, available to parties other than the sender and intended recipient(s), without formally interfering with channel confidentiality, that method violates the understood expectation of that security property.

4.4. Pattern inference is minimised

Analyses such as traffic fingerprinting or other (encrypted or unencrypted) data analysis techniques should be considered outside the scope of an E2EE system's goals of providing secure communications to end users.

Such methods of analyses, outside of or as part of E2EE system design, allow third parties to draw inferences from communication that was intended to be confidential. "By allowing private user data to be scanned via direct access by servers and their providers," the use of these methods should be considered an affront to "the privacy expectations of users of end-to-end encrypted communication systems" [GEC-EU].

Not only should an E2EE system value user data privacy by not enabling pattern inference, it should actively be attempting to solve issues of metadata and traceability (enhanced metadata) through further innovation that stays ahead of advances in these techniques.

4.5. The E2EE system is not compromised

RFC 3552 talks about the Internet Threat model such as the assumption that the user can expect any communications systems, but perhaps especially E2EE systems, to not be intentionally compromised [RFC3552]. Intentional compromises of E2EE systems are often referred to as "backdoors" but are often presented as additional design features under terms like "key escrow." Users of E2EE systems would not expect a front, back or side door entrance into their confidential conversations and would expect a provider to actively resist - technically and legally - compromise through these means.

5. Conclusions

From messaging to video conferencing, there are many competing features in an E2EE system that is secure and usable. The most well designed system cannot meet the expectations of every user, nor does an ideal system exist from any dimension. E2EE is a technology that is constantly improving to achieve the ideal as defined in this document.

Features and functionalities of E2EE systems should be developed and improved in service of end user expectations for privacy preserving communications.

6. Acknowledgements

Fred Baker, Stephen Farrell, Richard Barnes, Olaf Kolkman all contributed to the early strategic thinking of this document and whether it would be useful to the IETF community.

The folks at Riseup and the LEAP Encryption Access Project have articulated brilliantly the hardest parts of end-to-end encryption systems that serve the end users' right to whisper.

Ryan Polk at the Internet Society has energy to spare when it comes to organising meaningful contributions, like this one, for the technical advisors of the Global Encryption Coalition.

Adrian Farrel, are acknowledged for their review, comments, or questions that lead to improvement of this document.

7. Security Considerations

This document does not specify new protocols and therefore does not bring up technical security considerations.

Because some policy decisions may affect the security of the Internet, a clear and shared definition of end to end encrypted communication is important in policy related discussions. This document aims to provide that clarity.

8. IANA Considerations

This document has no actions for IANA.

9. Informative References

- [dkg] Gillmor, D., "Human Rights Protocol Considerations Glossary", 2015,
<<https://tools.ietf.org/html/draft-dkg-hrpc-glossary-00>>.
- [GEC-EU] Global Encryption Coalition, ., "Breaking encryption myths: What the European Commissions leaked report got wrong about online security", 2020,
<<https://www.globalencryption.org/2020/11/breaking-encryption-myths/>>.

- [komlo] Chelsea Komlo, ., "Defining end-to-end security", 2021,
<https://github.com/chelseakomlo/e2ee/blob/master/e2ee_definition.pdf>.
- [mls] IETF, ., "Messaging Layer Security", 2018,
<<https://datatracker.ietf.org/doc/charter-ietf-mls>>.
- [openpgp] IETF, ., "Open Specification for Pretty Good Privacy",
2020,
<<https://datatracker.ietf.org/doc/charter-ietf-openpgp>>.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC
Text on Security Considerations", BCP 72, RFC 3552,
DOI 10.17487/RFC3552, July 2003,
<<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC3724] Kempf, J., Ed., Austein, R., Ed., and IAB, "The Rise of
the Middle and the Future of End-to-End: Reflections on
the Evolution of the Internet Architecture", RFC 3724,
DOI 10.17487/RFC3724, March 2004,
<<https://www.rfc-editor.org/info/rfc3724>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an
Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May
2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T.,
Trammell, B., Huitema, C., and D. Borkmann,
"Confidentiality in the Face of Pervasive Surveillance: A
Threat Model and Problem Statement", RFC 7624,
DOI 10.17487/RFC7624, August 2015,
<<https://www.rfc-editor.org/info/rfc7624>>.
- [RFC8890] Nottingham, M., "The Internet is for End Users", RFC 8890,
DOI 10.17487/RFC8890, August 2020,
<<https://www.rfc-editor.org/info/rfc8890>>.
- [saltzer] Saltzer, et al, J.H., "End-to-end arguments in system
design", 1984,
<<https://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf>>.

Authors' Addresses

Mallory Knodel
CDT
Email: mknodel@cdt.org

Fred Baker
Email: fredbaker.IETF@gmail.com

Olaf Kolkman
ISOC
Email: kolkman@isoc.org

Sofía Celi
Cloudflare
Email: cherenkov@riseup.net

Gurshabad Grover
Centre for Internet and Society
Email: gurshabad@cis-india.org

sedispatch
Internet-Draft
Intended status: Standards Track
Expires: January 13, 2022

R. Struik
Struik Security Consultancy
July 12, 2021

ECDSA Signatures in Verification-Friendly Format
draft-struik-secdispatch-verify-friendly-ecdsa-00

Abstract

This document specifies how to represent ECDSA signatures so as to facilitate accelerated verification of single signatures and fast batch verification. We demonstrate that this representation technique can be applied retroactively by any device (rather than only by the signer), thereby facilitating transitioning to always generating ECDSA signatures in this way, without changing standardized ECDSA specifications with instantiations with prime-order curves. This facilitates verifying devices to reap the significant speed-up potential (ranging from ~1.3x to ~6x) fast verification techniques afford.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 13, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Fostering Fast Verification with ECDSA	2
2. Review of ECDSA and ECDSA*	3
3. Signature Verification with ECDSA and ECDSA*	4
4. Transitional Considerations	5
5. Implementation Status	6
6. Informal Comparison with Speed-ups for EdDSA Signatures . . .	6
7. Security Considerations	7
8. Privacy Considerations	7
9. IANA Considerations	7
9.1. OIDs for Use with PKIX and CMS	7
9.2. Algorithm Id for ECDSA* with OpenPGP	9
9.3. Other Uses	9
10. Acknowledgements	9
11. References	9
11.1. Normative References	10
11.2. Informative References	11
Author's Address	11

1. Fostering Fast Verification with ECDSA

ECDSA is one of the most widely used elliptic-curve digital signature algorithms. It has been standardized in FIPS Pub 186-4, ANSI X9.62, BSI, SECG, and IETF, and is widely deployed by a plethora of internet protocols specified by the Internet Engineering Task Force (IETF), with industry specifications in the areas of machine-to-machine communication, such as ZigBee, ISA, and Thread, with wireless communication protocols, such as IEEE 802.11, with payment protocols, such as EMV, with vehicle-to-vehicle (V2V) specifications, as well as with electronic travel documents and other specifications developed under a more stringent regulatory oversight regime, such as, e.g., ICAO and PIV. ECDSA is the only elliptic-curve based signature

scheme endorsed by regulatory bodies in both the United States and the European Union.

While methods for accelerated verification of ECDSA signatures and for combining this with key computations have been known for over 1 1/2 decade (see, e.g., [SAC2005] and [SAC2010]), these have been commonly described in technical papers in terms of ECDSA*, a slightly modified version of ECDSA, where their use with standardized ECDSA seems less well known. It is the purpose of this document to bridge this gap and describe how ECDSA signatures can be easily generated to facilitate more efficient verification, without failing. We emphasize that this does not require changes to standardized specifications of ECDSA instantiated with prime-order curves, thereby allowing reuse of existing standards and easy integration with existing implementations. We exemplify this for ECDSA certificates.

2. Review of ECDSA and ECDSA*

In this section, we summarize the properties of the signature scheme ECDSA and of the modified signature scheme ECDSA* that are relevant for our exposition (for more details, see, e.g., Appendix Q of [I-D.ietf-lwig-curve-representations]). The signature schemes are defined in terms of a suitable elliptic curve E , hash function H , and several representation functions, where n is the (prime) order of the base point G of this curve, and where E is an elliptic curve in short-Weierstrass form. For full details, we refer to the relevant standards.

With the ECDSA signature scheme, the signature over a message m provided by a signing entity with static private key d is an ordered pair (r,s) of integers in the interval $[1,n-1]$, where the value r is derived from a so-called ephemeral signing key $R:=k*G$ generated by the signer via a fixed public conversion function and where the value s is a function of the ephemeral private key k , the static private key d , the value r and the value e derived from message m via hash function H and representation hereof in the interval $[0,n-1]$. (More specifically, one has $e=s*k-d*r \pmod n$, where r is a function of the x-coordinate of R .) A signature (r,s) over message m purportedly signed by an entity with public key $Q:=d*G$ is accepted if Q is indeed a valid public key, if both signature components r and s are integers in the interval $[1,n-1]$ and if the reconstructed value R' derived from the purported signature, message, and public key yields r , via the same fixed conversion function as used during the signing operation. (More specifically, one computes $R':=(1/s)*(e*G+r*Q)$ and checks that r is the same function of the x-coordinate of R' .)

With the ECDSA* signature scheme, one follows the same signing operation, except that one outputs as signature the ordered pair

(R,s) , rather than the pair (r,s) , where R is the ephemeral signing key; one accepts a signature (R,s) over message m purportedly signed by an entity with public key Q by first computing the value r derived from signature component R via the conversion function, checking that Q is indeed a valid public key and that both r and s are integers in the interval $[1,n-1]$, computing $R' := (1/s) * (e * G + r * Q)$ and checking whether, indeed, $R' = R$.

It is known that ECDSA signatures and the corresponding ECDSA* signatures have the same success/failure conditions (i.e., ECDSA and ECDSA* are equally secure): if (r,s) is a valid ECDSA signature for message m purportedly signed by an entity with public key Q , then (R',s) is a valid corresponding ECDSA* signature, where $R' := (1/s) * (e * G + r * Q)$ is a point for which the conversion function yields r . Conversely, if (R,s) is a valid ECDSA* signature for message m purportedly signed by an entity with public key Q , then (r,s) is a valid corresponding ECDSA signature, where r is obtained from R via the conversion function.

It is well-known that if an ECDSA signature (r,s) is valid for a particular message m and public key Q , then so is $(r,-s)$ -- the so-called malleability -- and that, similarly, if an ECDSA* signature (R,s) is valid, then so is $(-R,-s)$, where this relies on the fact that the conversion function only depends on the x-coordinate of R .

3. Signature Verification with ECDSA and ECDSA*

In this section, we more closely scrutinize ECDSA and ECDSA* verification processes.

With ECDSA*, signature verification primarily involves checking an elliptic curve equation, viz. checking whether $R = (1/s) * (e * G + r * Q)$, which lends itself to accelerated signature verification techniques and the ability to use batch verification techniques, with significant potential for accelerated verification (with ~1.3x and up to ~6x speed-up potential, respectively). Here, speed-ups are due to the availability of the point R , which effectively allows checking an equation of the form $-s * R + (e * G + r * Q) = O$ instead (where O is the identity element of the curve). Similarly to the case with EdDSA [RFC8032] (which natively represents the ephemeral signing key R as part of the signature), this offers the potential for batch verification, by checking a randomized linear combination of this equation instead (thereby sharing the so-called point doubling operations amongst all individual verifications and, potentially, sharing scalars for signers of more than one message). In the case of single verifications, efficient tricks allow reducing the bit-size of the scalars involved in evaluating this expression (thereby effectively halving the required point doubling operations).

With ECDSA itself, these techniques are generally not available, since one cannot uniquely (and efficiently) reconstruct R from r : both R and $-R$ yield the same r value. If the conversion function only has two pre-images, though, one can use malleability to remove ambiguity altogether.

The modified ECDSA signing procedure is as follows:

- a. Generate ECDSA signature (r,s) of message m ;
- b. If the ephemeral signing key R has odd parity of the y -coordinate, change (r,s) to $(r,-s)$.

Note that this modified signing procedure removes the ambiguity in the reconstruction of R from r if the conversion function would otherwise only have two preimages, since R and $-R$ have different parity of the y -coordinate. In practice, this is the case for all prime-order curves, including the NIST prime curves P-256, P-384, P-521, all standardized Brainpool curves, and, e.g., the "BitCoin" curve secp256k1. (This follows from the observation that, for prime-order curves, r generally uniquely represents the x -coordinate of R .)

NOTE: With ECDSA, any party (not just the signer) can recompute the ephemeral signing key R' from a valid signature, since $R' := (1/s)(e*G + r*Q)$. In particular, any party can retroactively put the ECDSA signature in the required form above, thereby allowing subsequent unique reconstruction of the R value from r by verifying entities that know this modified signing procedure was indeed followed (again, subject to the assumption that r would only have two preimages otherwise, as is generally the case with prime-order curves).

One can extend this technique to also apply to curves that have a small co-factor h , e.g., $h=4$ or $h=8$ (rather than $h=1$, as is the case with prime-order curves). This extension is out of scope for the current document.

4. Transitional Considerations

The modified signing procedure described in Section 3 facilitates the use of accelerated ECDSA verification techniques by devices that wish to do so, provided these know that this modified signing procedure was indeed followed. This can be realized explicitly via a new "fast-verification-friendly" label (e.g., OID) indicating that this was indeed the case. This has the following consequences:

- a. New device: accept both old and new label and apply speed-ups with new label if possible (and desired);

- b. Old device: implement flimsy parser that replaces new label by old label and proceed as with traditional ECDSA verification.

Note that this parser "label replacement" step is a public operation, so any interface can implement this step.

A label can also be realized implicitly (e.g., by stipulating the modified signing procedure in protocol specifications that use ECDSA signatures), where the benefit of not having to introduce a new label explicitly should be weighed against potential disadvantages of implicit labels, such as requiring extra care with specification work to avoid confusion and the likely need to reintroduce an explicit label if ECDSA signatures are processed outside the original context (e.g., using a generic cryptographic token).

As suggested before, any device can implement the modified ECDSA signing procedure retroactively, so one could conceivably implement this once for all existing ECDSA signatures and only use "new" labels once this task has been completed (i.e., old labels could be mothballed from then on).

NOTE: the above labeling procedures assume that old and new labels are not part of the message to be signed. If they are, one may not be able to mothball old labels. In this case, signing devices should always use the old label during ECDSA signing and only change this to the corresponding new label afterwards, whereby verifying devices always replace the new label (since simply a pseudonym) by the corresponding old label before processing the ECDSA signature. This ensures that the signature semantics are not impacted and that old devices' ECDSA verification implementations (after reinstating old labels) work as is, while still being able to flag verification-friendly ECDSA signature formatting.

5. Implementation Status

[Note to the RFC Editor] Please remove this entire section before publication, as well as the reference to [RFC7942].

The ECDSA* signature scheme has been implemented in V2V specifications [P1609.2], where ECDSA is used with the NIST curves P-224 and P-256.

6. Informal Comparison with Speed-ups for EdDSA Signatures

The main message of this draft is as follows (no crypto required, except believing that the third step below works):

- a. EdDSA [RFC8032] does allow speedy signature verification and batch verification, since the signature is (R,s) , i.e., it represents the ephemeral signing key R as part of the signature;
- b. With ECDSA, the signature is (r,s) , where r is derived from the signing key R (essentially, r is the x-coordinate of R if the curve has co-factor $h=1$). However, generally, one cannot go back and get $(r,s) \rightarrow (R,s)$, at least not efficiently;
- c. If one uses the modified ECDSA signing procedure of Section 3, one can, though, thereby allowing similar accelerations (30% and up) for signature verification as EdDSA does. This can be viewed as "point compression" (since it determines which of R and $-R$ apply);
- d. The rest is detail, where the ideas underlying the speed-ups informally described in Section 3 are described in detail in the papers [SAC2005] and [SAC2010].

7. Security Considerations

The signature representation change described in this document is publicly known and, therefore, does not affect security provisions. Obviously, any adversary could change the signature value in a malicious way, so as to make signature verification fail. This does, however, not extend capabilities the adversary already had.

8. Privacy Considerations

The signature representation change described in this document is publicly known and, therefore, does not affect privacy provisions.

9. IANA Considerations

This section requests the following IANA code point assignments.

Editorial Note: the approach below is simply one way of realizing ECDSA* functionality. Other options to consider include, e.g., introducing a non-critical extension as label, where old devices can simply ignore this. This will be elaborated upon further in next versions of this draft, after feedback.

9.1. OIDs for Use with PKIX and CMS

This section registers the following object identifiers for the verification-friendly version of ECDSA introduced in this document:

- a. `id-ecdsa-star-with-sha256 ::= {iso(1) identified-organization(3) thawte (101) (100) 81};`
- b. `id-ecdsa-star-with-sha384 ::= {iso(1) identified-organization(3) thawte (101) (100) 82};`
- c. `id-ecdsa-star-with-sha512 ::= {iso(1) identified-organization(3) thawte (101) (100) 83};`
- d. `id-ecdsa-star-with-shake128 ::= {iso(1) identified-organization(3) thawte (101) (100) 84};`
- e. `id-ecdsa-star-with-shake256 ::= {iso(1) identified-organization(3) thawte (101) (100) 85}.`

Each of these object identifiers indicates the use of ECDSA with the indicated hash function, as the corresponding object identifiers without the "-star-" substring specified in [RFC5480] (for ECDSA with SHA2-hash family members) and in [RFC8692] (for ECDSA with SHAKE family members) do, where the "-star-" substring simply indicates that the modified signing procedure specified in Section 3 of this document was indeed used.

These new object identifiers are used with PKIX certificates and CMS in the same way as the corresponding object identifiers without the "-star-" substring, except that verifying devices now have the option to implement ECDSA signature verification as if ECDSA* signatures had been used, since the new object identifiers indicate the modified signing operation was followed, as illustrated in Section 3 of this document.

As mentioned in Section 4, any ECDSA signature with the old object identifier can be changed retroactively to one with the corresponding new object identifier, provided one has assurance that the modified ECDSA signing procedure was indeed followed and, conversely, any ECDSA signature with the new object identifier can be changed to one with the corresponding old object identifier, without change in semantics (assuming these object identifiers are not part of the message that is signed).

With [RFC5280], the signature algorithm is indicated twice: once as `signatureAlgorithm` field of the `Certificate` and once as the `Signature` field of the sequence `tbsCertificate`, where the former is not part of the message to be signed, whereas the latter is. Moreover, these two fields are stipulated to be the same (see Sections 4.1.1.2 and 4.1.2.3 of [RFC5280]). In this case, old and new labels **MUST** be used as indicated in the NOTE of Section 3, where the two fields indicating the signature algorithm are always both changed at the

same time (thereby, strictly complying with MUST behavior of PKIX that these two fields should be the same).

9.2. Algorithm Id for ECDSA* with OpenPGP

This section registers the ECDSA signature scheme with the modified signing procedure of this document as the public-key algorithm ECDSA* (with ID=25) in Section 9.1 of [I-D.ietf-openpgp-crypto-refresh] by including the following item in Table 15 of that section:

ID	Algorithm
25	ECDSA*

Table 1: Public-Key Algorithm Registry

As before, the provisions of Section 4 apply.

9.3. Other Uses

As suggested in Section 4, any party can retroactively put ECDSA signatures into the verification-friendly format, thereby conceivably allowing this to be done once and for all for all existing ECDSA signatures, no matter the application. In particular, one could apply this to ECDSA-based certificate chains, ECDSA-signed firmware updates, COSE, JOSE, etc., etc. In other words: going forward, never use ECDSA signing, always use ECDSA* signing.

Similar techniques can be used to put the German ECGDSA signature scheme, the Russian GOST signature scheme, and Chinese SM2 signature in a verification-friendly format, although this cannot be done retroactively without changing the signature format (it requires one extra bit). Further details are left to a future version of this document.

10. Acknowledgements

Thanks to Rich Salz for suggesting to informally compare speed-ups with ECDSA* with those of EdDSA (now in Section 6).

11. References

11.1. Normative References

[FIPS-186-4]

FIPS 186-4, "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication 186-4", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, July 2013.

[I-D.ietf-lwig-curve-representations]

Struik, R., "Alternative Elliptic Curve Representations", draft-ietf-lwig-curve-representations-20 (work in progress), February 2021.

[I-D.ietf-openpgp-crypto-refresh]

Koch, W. and P. Wouters, "OpenPGP Message Format", draft-ietf-openpgp-crypto-refresh-03 (work in progress), May 2021.

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

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

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <<https://www.rfc-editor.org/info/rfc5480>>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.

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

- [RFC8692] Kampanakis, P. and Q. Dang, "Internet X.509 Public Key Infrastructure: Additional Algorithm Identifiers for RSASSA-PSS and ECDSA Using SHAKEs", RFC 8692, DOI 10.17487/RFC8692, December 2019, <<https://www.rfc-editor.org/info/rfc8692>>.
- [SEC1] SEC1, "SEC 1: Elliptic Curve Cryptography, Version 2.0", Standards for Efficient Cryptography, , June 2009.
- [SEC2] SEC2, "SEC 2: Elliptic Curve Cryptography, Version 2.0", Standards for Efficient Cryptography, , January 2010.

11.2. Informative References

- [ECC] I.F. Blake, G. Seroussi, N.P. Smart, "Elliptic Curves in Cryptography", Cambridge University Press, Lecture Notes Series 265, July 1999.
- [GECC] D. Hankerson, A.J. Menezes, S.A. Vanstone, "Guide to Elliptic Curve Cryptography", New York: Springer-Verlag, 2004.
- [P1609.2] IEEE 1609.2-2013, "IEEE Standard for Wireless Access in Vehicular Environments-Security Services for Applications and Management Messages", IEEE Vehicular Technology Society, New York: IEEE, 2013.
- [SAC2005] A. Antipa, D.R. Brown, R. Gallant, R. Lambert, R. Struik, S.A. Vanstone, "Accelerated Verification of ECDSA Signatures", SAC 2005, B. Preneel, S. Tavares, Eds., Lecture Notes in Computer Science, Vol. 3897, pp. 307-318, Berlin: Springer, 2006.
- [SAC2010] R. Struik, "Batch Computations Revisited: Combining Key Computations and Batch Verifications", SAC 2010, A. Biryukov, G. Gong, D.R. Stinson, Eds., Lecture Notes in Computer Science, Vol. 6544, pp. 130-142, Berlin-Heidelberg: Springer, 2011.

Author's Address

Rene Struik
Struik Security Consultancy

Email: rstruik.ext@gmail.com

Internet Engineering Task Force
Internet-Draft
Updates: 6353 (if approved)
Intended status: Standards Track
Expires: 29 December 2021

K. Vaughn, Ed.
Trevilon LLC
27 June 2021

Transport Layer Security Version 1.3 (TLS 1.3) Transport Model for the
Simple Network Management Protocol Version 3 (SNMPv3)
draft-vaughn-tlstm-update-01

Abstract

This document updates the TLS Transport Model (TLSTM), as defined in [RFC6353], to support Transport Layer Security Version 1.3 (TLS) [RFC8446] and Datagram Transport Layer Security Version 1.3 (DTLS) [I-D.ietf-tls-dtls13], which are jointly known as "(D)TLS". This document may be applicable to future versions of SNMP and (D)TLS.

This document updates the SNMP-TLS-TM-MIB as defined in [RFC6353].

Status of This Memo

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

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

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

This Internet-Draft will expire on 29 December 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Conventions	3
2. Changes from RFC 6353	4
2.1. TLSTM Fingerprint	4
2.2. Security Level	5
2.3. TLS Version	5
2.4. SNMP Version	5
2.5. Common Name	6
3. Additional Rules for TLS 1.3	6
3.1. Zero Round Trip Time Resumption (0-RTT)	6
3.2. TLS ciphersuites, extensions and protocol invariants	6
4. MIB Module Definition	6
5. Security Considerations	37
5.1. MIB Module Security	37
6. IANA Considerations	38
7. Acknowledgements	39
8. References	39
8.1. Normative References	39
8.2. Informative References	40
Appendix A. Target and Notification Configuration Example	41
A.1. Configuring a Notification Originator	41
A.2. Configuring TLSTM to Utilize a Simple Derivation of tmSecurityName	42
A.3. Configuring TLSTM to Utilize Table-Driven Certificate Mapping	42
Author's Address	43

1. Introduction

This document updates the fingerprint algorithm defined by [RFC6353] to support the ciphersuites used by Transport Layer Security Version 1.3 (TLS) and Datagram Transport Layer Security Version 1.3 (DTLS), which are jointly known as "(D)TLS". The update also incorporates other less critical updates. Although the title and text of this document specifically reference SNMPv3 and (D)TLS 1.3, this document may be applicable to future versions of these protocols.

1.1. Conventions

Within this document the terms "TLS", "DTLS", "(D)TLS", "SNMP", and "TLSTM" mean "TLS 1.3", "DTLS 1.3", "TLS 1.3 and/or DTLS 1.3", "SMNPv3", and "TLSTM 1.3", respectively. These version numbers are only used when the text needs to emphasize version numbers, such as within the title. When this document refers to any other version of these protocols, it always explicitly states the version intended.

For consistency with SNMP-related specifications, this document favors terminology as defined in [STD62], rather than favoring terminology that is consistent with non-SNMP specifications. This is consistent with the IESG decision to not require the SNMPv3 terminology be modified to match the usage of other non-SNMP specifications when SNMPv3 was advanced to a Full Standard.

"Authentication" in this document typically refers to the English meaning of "serving to prove the authenticity of" the message, not data source authentication or peer identity authentication. The terms "manager" and "agent" are not used in this document because, in the RFC3411 architecture, all SNMP entities have the capability of acting as manager, agent, or both depending on the SNMP application types supported in the implementation. Where distinction is necessary, the application names of command generator, command responder, notification originator, notification receiver, and proxy forwarder are used. See "SNMP Applications" (RFC3411) for further information.

Throughout this document, the terms "client" and "server" are used to refer to the two ends of the TLS transport connection. The client actively opens the TLS connection, and the server passively listens for the incoming TLS connection. An SNMP entity MAY act as a TLS client or server or both, depending on the SNMP applications supported.

While TLS frequently refers to a user, the terminology preferred in RFC3411 and in this memo is "principal". A principal is the "who" on whose behalf services are provided or processing takes place. A principal can be, among other things, an individual acting in a particular role; a set of individuals, with each acting in a particular role; an application or a set of applications, or a combination of these within an administrative domain.

Throughout this document, the term "session" is used to refer to a secure association between two TLS Transport Models that permits the transmission of one or more SNMP messages within the lifetime of the session. The TLS protocol also has an internal notion of a session and although these two concepts of a session are related, when the term "session" is used this document is referring to the TLSTM's specific session and not directly to the TLS protocol's session.

The User-Based Security Model (USM) (RFC3414) is a mandatory-to-implement Security Model in [STD62]. The USM derives the securityName and securityLevel from the SNMP message received, even when the message was received over a secure transport. It is RECOMMENDED that deployments that support the TLSTM disable the USM, if it has been implemented.

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. Changes from RFC 6353

This document updates [RFC6353]. The changes from [RFC6353] are defined in the following clauses.

2.1. TLSTM Fingerprint

[RFC6353] defines a fingerprint algorithm that references the one-octet TLS 1.2 hash algorithm identifier. TLS 1.3 replaced the one-octet hash algorithm identifier with a two-octet TLS 1.3 cipher suite identifier thereby breaking the algorithm defined in [RFC6353]. The update to the SNMP-TLS-TM-MIB, as defined in Section 4, deprecates the original fingerprint TEXTUAL-CONVENTION and replaces it with a new TEXTUAL-CONVENTION.

The change also required an update to several objects within the tables defined within the SNMP-TLS-TM-MIB; further these objects are referenced by other (e.g., RowStatus) objects in a manner that requires deprecating and replacing the tables in their entirety. Thus, while the number of objects deprecated and replaced is significant the semantics of the changes are minor.

References to the older objects within [RFC6353] are applicable to the replacement objects. The newer objects are identified with names similar to those used in the original MIB but with a "13" inserted to reference TLS 1.3.

2.2. Security Level

The RFC3411 architecture recognizes three levels of security:

- * without authentication and without privacy (noAuthNoPriv)
- * with authentication but without privacy (authNoPriv)
- * with authentication and with privacy (authPriv)

With (D)TLS 1.3, authentication and privacy are always provided. Hence, all exchanges conforming to the rules of this document will include authentication and privacy, regardless of the security level requested.

```
// This is consistent with what was prescribed in RFC6353, where a
// TLS Transport Model is expected to provide for outgoing
// connections with a security level at least that of the requested
// security level.
```

2.3. TLS Version

[RFC6353] stated that TLSTM clients and servers MUST NOT request, offer, or use SSL 2.0. This document extends this statement such that TLSTM clients and servers MUST NOT request, offer, or use SSL 3.0, (D)TLSv 1.0, (D)TLS v1.1. See Appendix D.5 of [RFC8446] for further details. For backward compatibility issues with older TLS versions, see Appendix D of [RFC8446].

An implementation that supports these older protocols is not considered conformant to the TLSTM while the older protocols are enabled.

2.4. SNMP Version

[RFC6353] stated that using a non-transport-aware Security Model with a secure Transport Model was not recommended. This document tightens this statement such that TLSTM clients and servers MUST NOT request, offer, or use SNMPv1 or SNMPv2c message processing described in [RFC3584], or the User-based Security Model of SNMPv3.

An implementation that supports these older protocols is not considered conformant to the TLSTM while the older protocols are enabled.

2.5. Common Name

[RFC6353] stated that the use of a certificate's CommonName is deprecated and users were encouraged to use the subjectAltName. This document tightens this statement such that TLSTM clients and servers MUST NOT use the CommonName.

3. Additional Rules for TLS 1.3

This document specifies additional rules and clarifications for the use of TLS 1.3.

3.1. Zero Round Trip Time Resumption (0-RTT)

TLS 1.3 implementations for SNMPv3 MUST NOT enable the 0-RTT mode of session resumption (either sending or accepting) and MUST NOT automatically resend 0-RTT data if it is rejected by the server. The reason 0-RTT is disallowed is that there are no "safe" messages that if replayed will be guaranteed to cause no harm at a server side: all incoming notification or command responses are meant to be acted upon only once. See Security considerations section for further details.

TLS TM clients and servers MUST NOT request, offer or use the 0-RTT mode of TLS 1.3. [RFC8446] removed the renegotiation supported in TLS 1.2 [RFC5246]; for session resumption, it introduced a zero-RTT (0-RTT) mode, saving a round-trip at connection setup at the cost of increased risk of replay attacks (it is possible for servers to guard against this attack by keeping track of all the messages received). [RFC8446] requires a profile be written for any application that wants to use 0-RTT, specifying which messages are "safe to use" on this mode. The reason 0-RTT is disallowed here is that there are no "safe" SNMPv3 messages that if replayed will be sure to cause no harm at a server side: all incoming notification or command responses have consequences and are to be acted upon only once.

Renegotiation of sessions is not supported as it is not supported by TLS 1.3.

3.2. TLS ciphersuites, extensions and protocol invariants

[RFC8446] section 9 requires that, in the absence of application profiles, certain cipher suites, TLS extensions, and TLS protocol invariants are mandatory to implement. This document does not specify an application profile, hence all of the compliance requirements in [RFC8446] apply.

4. MIB Module Definition

```
SNMP-TLS-TM-MIB DEFINITIONS ::= BEGIN
IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    OBJECT-IDENTITY, mib-2, snmpDomains,
    Counter32, Unsigned32, Gauge32, NOTIFICATION-TYPE
        FROM SNMPv2-SMI
        -- RFC 2578 or any update thereof
    TEXTUAL-CONVENTION, TimeStamp, RowStatus, StorageType,
    AutonomousType
        FROM SNMPv2-TC
        -- RFC 2579 or any update thereof
    MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
        FROM SNMPv2-CONF
        -- RFC 2580 or any update thereof
    SnmpAdminString
        FROM SNMP-FRAMEWORK-MIB
        -- RFC 3411 or any update thereof
    snmpTargetParamsName, snmpTargetAddrName
        FROM SNMP-TARGET-MIB
        -- RFC 3413 or any update thereof
;
snmpTlstmMIB MODULE-IDENTITY
    LAST-UPDATED "202106220000Z"

    ORGANIZATION "ISMS Working Group"
    CONTACT-INFO "Kenneth Vaughn
        Trevilon LLC
        6606 FM 1488 RD, STE 503
        Magnolia, TX 77354
        USA
        kvaughn@trevilon.com"

    DESCRIPTION "
        The TLS Transport Model MIB
        Copyright (c) 2010-2021 IETF Trust and the persons identified
        as authors of the code. All rights reserved.
        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info)."
    REVISION      "202106220000Z"
    DESCRIPTION   "This version of this MIB module is part of
        RFC XXXX; see the RFC itself for full legal
        notices. This version updated the MIB to
        support (D)TLS 1.3."

    REVISION      "201107190000Z"
    DESCRIPTION   "This version of this MIB module is part of
        RFC 6353; see the RFC itself for full legal
        notices. The only change was to introduce
        new wording to reflect require changes for
        IDNA addresses in the SnmpTLSAddress TC."
```

```

    REVISION      "201005070000Z"
    DESCRIPTION   "This version of this MIB module is part of
                  RFC 5953; see the RFC itself for full legal
                  notices."
    ::= { mib-2 198 }
-- *****
-- subtrees of the SNMP-TLS-TM-MIB
-- *****
snmpTlstmNotifications OBJECT IDENTIFIER ::= { snmpTlstmMIB 0 }
snmpTlstmIdentities     OBJECT IDENTIFIER ::= { snmpTlstmMIB 1 }
snmpTlstmObjects        OBJECT IDENTIFIER ::= { snmpTlstmMIB 2 }
snmpTlstmConformance    OBJECT IDENTIFIER ::= { snmpTlstmMIB 3 }
-- *****
-- snmpTlstmObjects - Objects
-- *****
snmpTLSTCPDomain OBJECT-IDENTITY
    STATUS      current
    DESCRIPTION
        "The SNMP over TLS via TCP transport domain. The
        corresponding transport address is of type SnmpTLSAddress.
        The securityName prefix to be associated with the
        snmpTLSTCPDomain is 'tls'. This prefix may be used by
        security models or other components to identify which secure
        transport infrastructure authenticated a securityName."
    REFERENCE
        "RFC 2579: Textual Conventions for SMIV2"
    ::= { snmpDomains 8 }
snmpDTLSUDPDDomain OBJECT-IDENTITY
    STATUS      deprecated
    DESCRIPTION
        "The SNMP over DTLS via UDP transport domain. The
        corresponding transport address is of type SnmpTLSAddress.
        The securityName prefix to be associated with the
        snmpDTLSUDPDDomain is 'dtls'. This prefix may be used by
        security models or other components to identify which secure
        transport infrastructure authenticated a securityName."
    REFERENCE
        "RFC 2579: Textual Conventions for SMIV2"
    ::= { snmpDomains 9 }
SnmpTLSAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1a"
    STATUS      current
    DESCRIPTION
        "Represents an IPv4 address, an IPv6 address, or a
        US-ASCII-encoded hostname and port number.
        An IPv4 address must be in dotted decimal format followed by a
        colon ':' (US-ASCII character 0x3A) and a decimal port number
        in US-ASCII."

```

An IPv6 address must be a colon-separated format (as described in RFC 5952), surrounded by square brackets ('[', US-ASCII character 0x5B, and ']', US-ASCII character 0x5D), followed by a colon ':' (US-ASCII character 0x3A) and a decimal port number in US-ASCII.

A hostname is always in US-ASCII (as per RFC 1123); internationalized hostnames are encoded as A-labels as specified in RFC 5890. The hostname is followed by a colon ':' (US-ASCII character 0x3A) and a decimal port number in US-ASCII. The name SHOULD be fully qualified whenever possible.

Values of this textual convention may not be directly usable as transport-layer addressing information, and may require run-time resolution. As such, applications that write them must be prepared for handling errors if such values are not supported, or cannot be resolved (if resolution occurs at the time of the management operation).

The DESCRIPTION clause of TransportAddress objects that may have SnmpTLSAddress values must fully describe how (and when) such names are to be resolved to IP addresses and vice versa.

This textual convention SHOULD NOT be used directly in object definitions since it restricts addresses to a specific format. However, if it is used, it MAY be used either on its own or in conjunction with TransportAddressType or TransportDomain as a pair.

When this textual convention is used as a syntax of an index object, there may be issues with the limit of 128 sub-identifiers specified in SMIV2 (STD 58). It is RECOMMENDED that all MIB documents using this textual convention make explicit any limitations on index component lengths that management software must observe. This may be done either by including SIZE constraints on the index components or by specifying applicable constraints in the conceptual row DESCRIPTION clause or in the surrounding documentation."

REFERENCE

- "RFC 1123: Requirements for Internet Hosts - Application and Support
- RFC 5890: Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework
- RFC 5952: A Recommendation for IPv6 Address Text Representation

SYNTAX OCTET STRING (SIZE (1..255))
SnmpTLSPFingerprint ::= TEXTUAL-CONVENTION
DISPLAY-HINT "1x:1x"
STATUS deprecated
DESCRIPTION

"A fingerprint value that can be used to uniquely reference other data of potentially arbitrary length.
 An SnmpTLSFingerprint value is composed of a 1-octet hashing algorithm identifier followed by the fingerprint value. The octet value encoded is taken from the IANA TLS HashAlgorithm Registry (RFC 5246). The remaining octets are filled using the results of the hashing algorithm.
 This TEXTUAL-CONVENTION allows for a zero-length (blank) SnmpTLSFingerprint value for use in tables where the fingerprint value may be optional. MIB definitions or implementations may refuse to accept a zero-length value as appropriate.
 This textual convention was deprecated because TLS 1.3 uses a 2-octet cipher suite identifier rather than a 1-octet hashing algorithm identifier."

REFERENCE "RFC 5246: The Transport Layer
 Security (TLS) Protocol Version 1.2
<http://www.iana.org/assignments/tls-parameters/>
 "

SYNTAX OCTET STRING (SIZE (0..255))
 SnmpTLS13Fingerprint ::= TEXTUAL-CONVENTION
 DISPLAY-HINT "1x,1x"
 STATUS current
 DESCRIPTION
 "A fingerprint value that can be used to uniquely reference other data of potentially arbitrary length.
 An SnmpTLS13Fingerprint value is composed of a 2-octet cipher suite identifier followed by the fingerprint value. The octet value encoded is taken from the IANA TLS Cipher Suites Registry (RFC 8446). The remaining octets are filled using the results of the hashing algorithm, up to the first 253 octets.
 This TEXTUAL-CONVENTION allows for a zero-length (blank) SnmpTLS13Fingerprint value for use in tables where the fingerprint value may be optional. MIB definitions or implementations may refuse to accept a zero-length value as appropriate."
 REFERENCE "RFC 8446: The Transport Layer
 Security (TLS) Protocol Version 1.3
<http://www.iana.org/assignments/tls-parameters/>
 "

SYNTAX OCTET STRING (SIZE (0..255))
 -- Identities for use in the snmpTlstmCertToTSNTable and
 -- snmpTlstmCertToTSN13Table
 snmpTlstmCertToTSNMIdentities OBJECT IDENTIFIER
 ::= { snmpTlstmIdentities 1 }
 snmpTlstmCertSpecified OBJECT-IDENTITY
 STATUS current
 DESCRIPTION "Directly specifies the tmSecurityName to be used for

this certificate. The value of the tmSecurityName to use is specified in the snmpTlstmCertToTSN13Data column. The snmpTlstmCertToTSN13Data column must contain a non-zero length SnmpAdminString compliant value or the mapping described in this row must be considered a failure."

```

 ::= { snmpTlstmCertToTSNMIdentities 1 }
snmpTlstmCertSANRFC822Name OBJECT-IDENTITY
STATUS          current
DESCRIPTION     "Maps a subjectAltName's rfc822Name to a
tmSecurityName. The local part of the rfc822Name is
passed unaltered but the host-part of the name must
be passed in lowercase. This mapping results in a
1:1 correspondence between equivalent subjectAltName
rfc822Name values and tmSecurityName values except
that the host-part of the name MUST be passed in
lowercase.
Example rfc822Name Field:  FooBar@Example.COM
is mapped to tmSecurityName: FooBar@example.com."

 ::= { snmpTlstmCertToTSNMIdentities 2 }
snmpTlstmCertSANDNSName OBJECT-IDENTITY
STATUS          current
DESCRIPTION     "Maps a subjectAltName's dNSName to a
tmSecurityName after first converting it to all
lowercase (RFC 5280 does not specify converting to
lowercase so this involves an extra step). This
mapping results in a 1:1 correspondence between
subjectAltName dNSName values and the tmSecurityName
values."

REFERENCE "RFC 5280 - Internet X.509 Public Key Infrastructure
Certificate and Certificate Revocation
List (CRL) Profile."

 ::= { snmpTlstmCertToTSNMIdentities 3 }
snmpTlstmCertSANIpAddress OBJECT-IDENTITY
STATUS          current
DESCRIPTION     "Maps a subjectAltName's ipAddress to a
tmSecurityName by transforming the binary encoded
address as follows:
1) for IPv4, the value is converted into a
decimal-dotted quad address (e.g., '192.0.2.1').
2) for IPv6 addresses, the value is converted into a
32-character all lowercase hexadecimal string
without any colon separators.
This mapping results in a 1:1 correspondence between
subjectAltName ipAddress values and the
tmSecurityName values.
The resulting length of an encoded IPv6 address is
the maximum length supported by the View-Based

```

Access Control Model (VACM). Using both the Transport Security Model's support for transport prefixes (see the SNMP-TSM-MIB's `snmpTsmConfigurationUsePrefix` object for details) will result in `securityName` lengths that exceed what VACM can handle."

```
 ::= { snmpTlstmCertToTSNMIdentities 4 }
snmpTlstmCertSANAny OBJECT-IDENTITY
STATUS      current
DESCRIPTION "Maps any of the following fields using the
             corresponding mapping algorithms:
```

Type	Algorithm
<code>rfc822Name</code>	<code>snmpTlstmCertSANRFC822Name</code>
<code>dNSName</code>	<code>snmpTlstmCertSANDNSName</code>
<code>iPAddress</code>	<code>snmpTlstmCertSANIpAddress</code>

The first matching `subjectAltName` value found in the certificate of the above types MUST be used when deriving the `tmSecurityName`. The mapping algorithm specified in the 'Algorithm' column MUST be used to derive the `tmSecurityName`.

This mapping results in a 1:1 correspondence between `subjectAltName` values and `tmSecurityName` values. The three sub-mapping algorithms produced by this combined algorithm cannot produce conflicting results between themselves."

```
 ::= { snmpTlstmCertToTSNMIdentities 5 }
snmpTlstmCertCommonName OBJECT-IDENTITY
STATUS      deprecated
DESCRIPTION "Maps a certificate's CommonName to a tmSecurityName
             after converting it to a UTF-8 encoding. The usage
             of CommonNames is deprecated and users are
             encouraged to use subjectAltName mapping methods
             instead. This mapping results in a 1:1
             correspondence between certificate CommonName values
             and tmSecurityName values."
 ::= { snmpTlstmCertToTSNMIdentities 6 }
```

```
-- The snmpTlstmSession Group
snmpTlstmSession      OBJECT IDENTIFIER ::= { snmpTlstmObjects 1 }
snmpTlstmSessionOpens OBJECT-TYPE
SYNTAX                Counter32
MAX-ACCESS             read-only
STATUS                 current
DESCRIPTION            "The number of times an openSession() request has been executed
```



```
    as a (D)TLS client, regardless of whether it succeeded or
    failed."
 ::= { snmpTlstmSession 1 }
snmpTlstmSessionClientCloses OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of times a closeSession() request has been
    executed as a (D)TLS client, regardless of whether it
    succeeded or failed."
 ::= { snmpTlstmSession 2 }
snmpTlstmSessionOpenErrors OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of times an openSession() request failed to open a
    session as a (D)TLS client, for any reason."
 ::= { snmpTlstmSession 3 }
snmpTlstmSessionAccepts OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of times a (D)TLS server has accepted a new
    connection from a client and has received at least one SNMP
    message through it."
 ::= { snmpTlstmSession 4 }

snmpTlstmSessionServerCloses OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of times a closeSession() request has been
    executed as a (D)TLS server, regardless of whether it
    succeeded or failed."
 ::= { snmpTlstmSession 5 }
snmpTlstmSessionNoSessions OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of times an outgoing message was dropped because
    the session associated with the passed tmStateReference was no
    longer (or was never) available."
 ::= { snmpTlstmSession 6 }
```

```
snmpTlstmSessionInvalidClientCertificates OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an incoming session was not established
        on a (D)TLS server because the presented client certificate
        was invalid. Reasons for invalidation include, but are not
        limited to, cryptographic validation failures or lack of a
        suitable mapping row in the snmpTlstmCertToTSNTable or the
        snmpTlstmCertToTSN13Table."
    ::= { snmpTlstmSession 7 }
snmpTlstmSessionUnknownServerCertificate OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an outgoing session was not established
        on a (D)TLS client because the server certificate presented
        by an SNMP over (D)TLS server was invalid because no
        configured fingerprint or Certification Authority (CA) was
        acceptable to validate it.
        This may result because there was no entry in the
        snmpTlstmAddrTable (or snmpTlstmAddr13Table) or because no
        path could be found to a known CA."
    ::= { snmpTlstmSession 8 }
snmpTlstmSessionInvalidServerCertificates OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of times an outgoing session was not established
        on a (D)TLS client because the server certificate presented
        by an SNMP over (D)TLS server could not be validated even if
        the fingerprint or expected validation path was known. That
        is, a cryptographic validation error occurred during
        certificate validation processing.
        Reasons for invalidation include, but are not
        limited to, cryptographic validation failures."
    ::= { snmpTlstmSession 9 }
snmpTlstmSessionInvalidCaches OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The number of outgoing messages dropped because the
        tmStateReference referred to an invalid cache."
    ::= { snmpTlstmSession 10 }
```

```

-- Configuration Objects
snmpTlstmConfig          OBJECT IDENTIFIER ::= {snmpTlstmObjects 2}
-- Certificate mapping
snmpTlstmCertificateMapping OBJECT IDENTIFIER ::= {snmpTlstmConfig 1}
snmpTlstmCertToTSNCount OBJECT-TYPE
    SYNTAX          Gauge32
    MAX-ACCESS      read-only
    STATUS           deprecated
    DESCRIPTION
        "A count of the number of entries in the
        snmpTlstmCertToTSNTable."
    ::= { snmpTlstmCertificateMapping 1 }
snmpTlstmCertToTSNTableLastChanged OBJECT-TYPE
    SYNTAX          TimeStamp
    MAX-ACCESS      read-only
    STATUS           deprecated
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmCertToTSNTable was
        last modified through any means, or 0 if it has not been
        modified since the command responder was started."
    ::= { snmpTlstmCertificateMapping 2 }
snmpTlstmCertToTSNTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF SnmpTlstmCertToTSNEntry
    MAX-ACCESS      not-accessible
    STATUS           deprecated
    DESCRIPTION
        "This table is used by a (D)TLS server to map the (D)TLS
        client's presented X.509 certificate to a tmSecurityName.
        On an incoming (D)TLS/SNMP connection, the client's presented
        certificate must either be validated based on an established
        trust anchor, or it must directly match a fingerprint in this
        table. This table does not provide any mechanisms for
        configuring the trust anchors; the transfer of any needed
        trusted certificates for path validation is expected to occur
        through an out-of-band transfer.
        Once the certificate has been found acceptable (either by path
        validation or directly matching a fingerprint in this table),
        this table is consulted to determine the appropriate
        tmSecurityName to identify with the remote connection. This
        is done by considering each active row from this table in
        prioritized order according to its snmpTlstmCertToTSNID value.
        Each row's snmpTlstmCertToTSNFingerprint value determines
        whether the row is a match for the incoming connection:
        1) If the row's snmpTlstmCertToTSNFingerprint value
           identifies the presented certificate, then consider the
           row as a successful match.
        2) If the row's snmpTlstmCertToTSNFingerprint value
           identifies a locally held copy of a trusted CA

```

certificate and that CA certificate was used to validate the path to the presented certificate, then consider the row as a successful match.

Once a matching row has been found, the `snmpTlstmCertToTSNMapType` value can be used to determine how the `tmSecurityName` to associate with the session should be determined. See the `snmpTlstmCertToTSNMapType` column's DESCRIPTION for details on determining the `tmSecurityName` value. If it is impossible to determine a `tmSecurityName` from the row's data combined with the data presented in the certificate, then additional rows MUST be searched looking for another potential match. If a resulting `tmSecurityName` mapped from a given row is not compatible with the needed requirements of a `tmSecurityName` (e.g., VACM imposes a 32-octet-maximum length and the certificate derived `securityName` could be longer), then it must be considered an invalid match and additional rows MUST be searched looking for another potential match.

If no matching and valid row can be found, the connection MUST be closed and SNMP messages MUST NOT be accepted over it.

Missing values of `snmpTlstmCertToTSNID` are acceptable and implementations should continue to the next highest numbered row. It is recommended that administrators skip index values to leave room for the insertion of future rows (for example, use values of 10 and 20 when creating initial rows).

Users are encouraged to make use of certificates with `subjectAltName` fields that can be used as `tmSecurityNames` so that a single root CA certificate can allow all child certificate's `subjectAltName` to map directly to a `tmSecurityName` via a 1:1 transformation. However, this table is flexible to allow for situations where existing deployed certificate infrastructures do not provide adequate `subjectAltName` values for use as `tmSecurityNames`.

Direct mapping from each individual certificate fingerprint to a `tmSecurityName` is also possible but requires one entry in the table per `tmSecurityName` and requires more management operations to completely configure a device.

This table and its associated objects were deprecated because the fingerprint format changed to support TLS 1.3. By deprecating (and creating an updated) table, rather than just the fingerprint object, an implementation is able to support both the original TLS and new TLS 1.3 tables while forcing some agents to only use TLS 1.3."

```
 ::= { snmpTlstmCertificateMapping 3 }
snmpTlstmCertToTSNEntry OBJECT-TYPE
    SYNTAX      SnmpTlstmCertToTSNEntry
    MAX-ACCESS  not-accessible
```

```

STATUS      deprecated
DESCRIPTION
    "A row in the snmpTlstmCertToTSNTable that specifies a mapping
    for an incoming (D)TLS certificate to a tmSecurityName to use
    for a connection."
INDEX      { snmpTlstmCertToTSNID }
::= { snmpTlstmCertToTSNTable 1 }
snmpTlstmCertToTSNEntry ::= SEQUENCE {
    snmpTlstmCertToTSNID      Unsigned32,
    snmpTlstmCertToTSNFingerprint  SnmpTLSFingerprint,
    snmpTlstmCertToTSNMapType  AutonomousType,
    snmpTlstmCertToTSNData      OCTET STRING,
    snmpTlstmCertToTSNStorageType  StorageType,
    snmpTlstmCertToTSNRowStatus  RowStatus
}
snmpTlstmCertToTSNID OBJECT-TYPE
SYNTAX      Unsigned32 (1..4294967295)
MAX-ACCESS  not-accessible
STATUS      deprecated
DESCRIPTION
    "A unique, prioritized index for the given entry.  Lower
    numbers indicate a higher priority."
::= { snmpTlstmCertToTSNEntry 1 }
snmpTlstmCertToTSNFingerprint OBJECT-TYPE
SYNTAX      SnmpTLSFingerprint (SIZE(1..255))
MAX-ACCESS  read-create
STATUS      deprecated
DESCRIPTION
    "A cryptographic hash of an X.509 certificate.  The results of
    a successful matching fingerprint to either the trusted CA in
    the certificate validation path or to the certificate itself
    is dictated by the snmpTlstmCertToTSNMapType column.
    This object was deprecated because TLS 1.3 uses a 2-octet
    cipher suite identifier rather than a 1-octet hashing algorithm
    identifier."
::= { snmpTlstmCertToTSNEntry 2 }
snmpTlstmCertToTSNMapType OBJECT-TYPE
SYNTAX      AutonomousType
MAX-ACCESS  read-create
STATUS      deprecated
DESCRIPTION
    "Specifies the mapping type for deriving a tmSecurityName from
    a certificate.  Details for mapping of a particular type SHALL
    be specified in the DESCRIPTION clause of the OBJECT-IDENTITY
    that describes the mapping.  If a mapping succeeds it will
    return a tmSecurityName for use by the TLSTM model and
    processing stops.
    If the resulting mapped value is not compatible with the

```

needed requirements of a tmSecurityName (e.g., VACM imposes a 32-octet-maximum length and the certificate derived securityName could be longer), then future rows MUST be searched for additional snmpTlstmCertToTSNFingerprint matches to look for a mapping that succeeds.

Suitable values for assigning to this object that are defined within the SNMP-TLS-TM-MIB can be found in the snmpTlstmCertToTSNMIdentities portion of the MIB tree."

```

DEFVAL { snmpTlstmCertSpecified }
::= { snmpTlstmCertToTSNEntry 3 }
snmpTlstmCertToTSNData OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE(0..1024))
    MAX-ACCESS   read-create
    STATUS       deprecated
    DESCRIPTION
        "Auxiliary data used as optional configuration information for
        a given mapping specified by the snmpTlstmCertToTSNMapType
        column. Only some mapping systems will make use of this
        column. The value in this column MUST be ignored for any
        mapping type that does not require data present in this
        column."
    DEFVAL { "" }
    ::= { snmpTlstmCertToTSNEntry 4 }
snmpTlstmCertToTSNStorageType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS   read-create
    STATUS       deprecated
    DESCRIPTION
        "The storage type for this conceptual row. Conceptual rows
        having the value 'permanent' need not allow write-access to
        any columnar objects in the row."
    DEFVAL      { nonVolatile }
    ::= { snmpTlstmCertToTSNEntry 5 }
snmpTlstmCertToTSNRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS   read-create
    STATUS       deprecated
    DESCRIPTION
        "The status of this conceptual row. This object may be used
        to create or remove rows from this table.
        To create a row in this table, an administrator must set this
        object to either createAndGo(4) or createAndWait(5).
        Until instances of all corresponding columns are appropriately
        configured, the value of the corresponding instance of the
        snmpTlstmParamsRowStatus column is notReady(3).
        In particular, a newly created row cannot be made active until
        the corresponding snmpTlstmCertToTSNFingerprint,
        snmpTlstmCertToTSNMapType, and snmpTlstmCertToTSNData columns

```

have been set.
The following objects may not be modified while the value of this object is active(1):

- snmpTlstmCertToTSNFingerprint
- snmpTlstmCertToTSNMapType
- snmpTlstmCertToTSNData

An attempt to set these objects while the value of snmpTlstmParamsRowStatus is active(1) will result in an inconsistentValue error."

```
::= { snmpTlstmCertToTSNEntry 6 }
-- Maps tmSecurityNames to certificates for use by SNMP-TARGET-MIB
snmpTlstmParamsCount OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS   read-only
    STATUS       deprecated
    DESCRIPTION
        "A count of the number of entries in the snmpTlstmParamsTable."
    ::= { snmpTlstmCertificateMapping 4 }
snmpTlstmParamsTableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS   read-only
    STATUS       deprecated
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmParamsTable
         was last modified through any means, or 0 if it has not been
         modified since the command responder was started."
    ::= { snmpTlstmCertificateMapping 5 }
snmpTlstmParamsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpTlstmParamsEntry
    MAX-ACCESS   not-accessible
    STATUS       deprecated
    DESCRIPTION
        "This table is used by a (D)TLS client when a (D)TLS
         connection is being set up using an entry in the
         SNMP-TARGET-MIB. It extends the SNMP-TARGET-MIB's
         snmpTargetParamsTable with a fingerprint of a certificate to
         use when establishing such a (D)TLS connection."
    ::= { snmpTlstmCertificateMapping 6 }
snmpTlstmParamsEntry OBJECT-TYPE
    SYNTAX      SnmpTlstmParamsEntry
    MAX-ACCESS   not-accessible
    STATUS       deprecated
    DESCRIPTION
        "A conceptual row containing a fingerprint hash of a locally
         held certificate for a given snmpTargetParamsEntry. The
         values in this row should be ignored if the connection that
         needs to be established, as indicated by the SNMP-TARGET-MIB
         infrastructure, is not a certificate and TLS based
```

connection. The connection SHOULD NOT be established if the certificate fingerprint stored in this entry does not point to a valid locally held certificate or if it points to an unusable certificate (such as might happen when the certificate's expiration date has been reached)."

```

INDEX      { IMPLIED snmpTargetParamsName }
 ::= { snmpTlstmParamsTable 1 }
snmpTlstmParamsEntry ::= SEQUENCE {
    snmpTlstmParamsClientFingerprint SnmpTLSEFingerprint,
    snmpTlstmParamsStorageType        StorageType,
    snmpTlstmParamsRowStatus          RowStatus
}
snmpTlstmParamsClientFingerprint OBJECT-TYPE
SYNTAX      SnmpTLSEFingerprint
MAX-ACCESS  read-create
STATUS      deprecated
DESCRIPTION
    "This object stores the hash of the public portion of a
    locally held X.509 certificate. The X.509 certificate, its
    public key, and the corresponding private key will be used
    when initiating a TLS connection as a TLS client."
 ::= { snmpTlstmParamsEntry 1 }
snmpTlstmParamsStorageType OBJECT-TYPE
SYNTAX      StorageType
MAX-ACCESS  read-create
STATUS      deprecated
DESCRIPTION
    "The storage type for this conceptual row. Conceptual rows
    having the value 'permanent' need not allow write-access to
    any columnar objects in the row."
DEFVAL      { nonVolatile }
 ::= { snmpTlstmParamsEntry 2 }
snmpTlstmParamsRowStatus OBJECT-TYPE
SYNTAX      RowStatus
MAX-ACCESS  read-create
STATUS      deprecated
DESCRIPTION
    "The status of this conceptual row. This object may be used
    to create or remove rows from this table.
    To create a row in this table, an administrator must set this
    object to either createAndGo(4) or createAndWait(5).
    Until instances of all corresponding columns are appropriately
    configured, the value of the corresponding instance of the
    snmpTlstmParamsRowStatus column is notReady(3).
    In particular, a newly created row cannot be made active until
    the corresponding snmpTlstmParamsClientFingerprint column has
    been set.
    The snmpTlstmParamsClientFingerprint object may not be modified

```



```

        while the value of this object is active(1).
        An attempt to set these objects while the value of
        snmpTlstmParamsRowStatus is active(1) will result in
        an inconsistentValue error."
 ::= { snmpTlstmParamsEntry 3 }
mpTlstmAddrCount OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS   read-only
    STATUS       deprecated
    DESCRIPTION
        "A count of the number of entries in the snmpTlstmAddrTable."
 ::= { snmpTlstmCertificateMapping 7 }
snmpTlstmAddrTableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS   read-only
    STATUS       deprecated
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmAddrTable
        was last modified through any means, or 0 if it has not been
        modified since the command responder was started."
 ::= { snmpTlstmCertificateMapping 8 }
snmpTlstmAddrTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpTlstmAddrEntry
    MAX-ACCESS   not-accessible
    STATUS       deprecated
    DESCRIPTION
        "This table is used by a TLS client when a TLS
        connection is being set up using an entry in the
        SNMP-TARGET-MIB. It extends the SNMP-TARGET-MIB's
        snmpTargetAddrTable so that the client can verify that the
        correct server has been reached. This verification can use
        either a certificate fingerprint, or an identity
        authenticated via certification path validation.
        If there is an active row in this table corresponding to the
        entry in the SNMP-TARGET-MIB that was used to establish the
        connection, and the row's snmpTlstmAddrServerFingerprint
        column has non-empty value, then the server's presented
        certificate is compared with the
        snmpTlstmAddrServerFingerprint value (and the
        snmpTlstmAddrServerIdentity column is ignored). If the
        fingerprint matches, the verification has succeeded. If the
        fingerprint does not match, then the connection MUST be
        closed.
        If the server's presented certificate has passed
        certification path validation [RFC5280] to a configured
        trust anchor, and an active row exists with a zero-length
        snmpTlstmAddrServerFingerprint value, then the
        snmpTlstmAddrServerIdentity column contains the expected

```

host name. This expected host name is then compared against the server's certificate as follows:

- Implementations MUST support matching the expected host name against a `dnsName` in the `subjectAltName` extension field
- The '*' (ASCII 0x2a) wildcard character is allowed in the `dnsName` of the `subjectAltName` extension, but only as the left-most (least significant) DNS label in that value. This wildcard matches any left-most DNS label in the server name. That is, the subject `*.example.com` matches the server names `a.example.com` and `b.example.com`, but does not match `example.com` or `a.b.example.com`. Implementations MUST support wildcards in certificates as specified above, but MAY provide a configuration option to disable them.
- If the locally configured name is an internationalized domain name, conforming implementations MUST convert it to the ASCII Compatible Encoding (ACE) format for performing comparisons, as specified in Section 7 of [RFC5280].

If the expected host name fails these conditions then the connection MUST be closed.

If there is no row in this table corresponding to the entry in the SNMP-TARGET-MIB and the server can be authorized by another, implementation-dependent means, then the connection MAY still proceed."

```
 ::= { snmpTlstmCertificateMapping 9 }
snmpTlstmAddrEntry OBJECT-TYPE
    SYNTAX      SnmpTlstmAddrEntry
    MAX-ACCESS  not-accessible
    STATUS      deprecated
    DESCRIPTION
        "A conceptual row containing a copy of a certificate's
        fingerprint for a given snmpTargetAddrEntry. The values in
        this row should be ignored if the connection that needs to be
        established, as indicated by the SNMP-TARGET-MIB
        infrastructure, is not a TLS based connection. If an
        snmpTlstmAddrEntry exists for a given snmpTargetAddrEntry, then
        the presented server certificate MUST match or the connection
        MUST NOT be established. If a row in this table does not
        exist to match an snmpTargetAddrEntry row, then the connection
        SHOULD still proceed if some other certificate validation path
        algorithm (e.g., RFC 5280) can be used."
    INDEX       { IMPLIED snmpTargetAddrName }
 ::= { snmpTlstmAddrTable 1 }
SnmpTlstmAddrEntry ::= SEQUENCE {
    snmpTlstmAddrServerFingerprint    SnmpTLSPFingerprint,
    snmpTlstmAddrServerIdentity       SnmpAdminString,
    snmpTlstmAddrStorageType          StorageType,
```

```

        snmpTlstmAddrRowStatus          RowStatus
    }
snmpTlstmAddrServerFingerprint OBJECT-TYPE
    SYNTAX          SnmpTLSFingerprint
    MAX-ACCESS      read-create
    STATUS           deprecated
    DESCRIPTION
        "A cryptographic hash of a public X.509 certificate.  This
        object should store the hash of the public X.509 certificate
        that the remote server should present during the TLS
        connection setup.  The fingerprint of the presented
        certificate and this hash value MUST match exactly or the
        connection MUST NOT be established."
    DEFVAL { "" }
    ::= { snmpTlstmAddrEntry 1 }
snmpTlstmAddrServerIdentity OBJECT-TYPE
    SYNTAX          SnmpAdminString
    MAX-ACCESS      read-create
    STATUS           deprecated
    DESCRIPTION
        "The reference identity to check against the identity
        presented by the remote system."
    DEFVAL { "" }
    ::= { snmpTlstmAddrEntry 2 }
snmpTlstmAddrStorageType OBJECT-TYPE
    SYNTAX          StorageType
    MAX-ACCESS      read-create
    STATUS           deprecated
    DESCRIPTION
        "The storage type for this conceptual row.  Conceptual rows
        having the value 'permanent' need not allow write-access to
        any columnar objects in the row."
    DEFVAL          { nonVolatile }
    ::= { snmpTlstmAddrEntry 3 }
snmpTlstmAddrRowStatus OBJECT-TYPE
    SYNTAX          RowStatus
    MAX-ACCESS      read-create
    STATUS           deprecated
    DESCRIPTION
        "The status of this conceptual row.  This object may be used
        to create or remove rows from this table.
        To create a row in this table, an administrator must set this
        object to either createAndGo(4) or createAndWait(5).
        Until instances of all corresponding columns are
        appropriately configured, the value of the
        corresponding instance of the snmpTlstmAddrRowStatus
        column is notReady(3).
        In particular, a newly created row cannot be made active until

```

the corresponding snmpTlstmAddrServerFingerprint column has been set.

Rows MUST NOT be active if the snmpTlstmAddrServerFingerprint column is blank and the snmpTlstmAddrServerIdentity is set to '*' since this would insecurely accept any presented certificate.

The snmpTlstmAddrServerFingerprint object may not be modified while the value of this object is active(1).

An attempt to set these objects while the value of snmpTlstmAddrRowStatus is active(1) will result in an inconsistentValue error."

```
 ::= { snmpTlstmAddrEntry 4 }
snmpTlstmCertToTSN13Count OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the
        snmpTlstmCertToTSN13Table."
 ::= { snmpTlstmCertificateMapping 10 }
snmpTlstmCertToTSN13TableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmCertToTSN13Table
        was last modified through any means, or 0 if it has not been
        modified since the command responder was started."
 ::= { snmpTlstmCertificateMapping 11 }
snmpTlstmCertToTSN13Table OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpTlstmCertToTSN13Entry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table is used by a TLS 1.3 server to map the TLS 1.3
        client's presented X.509 certificate to a tmSecurityName.
        On an incoming TLS/SNMP connection, the client's presented
        certificate must either be validated based on an established
        trust anchor, or it must directly match a fingerprint in this
        table. This table does not provide any mechanisms for
        configuring the trust anchors; the transfer of any needed
        trusted certificates for path validation is expected to occur
        through an out-of-band transfer.
        Once the certificate has been found acceptable (either by path
        validation or directly matching a fingerprint in this table),
        this table is consulted to determine the appropriate
        tmSecurityName to identify with the remote connection. This
```

is done by considering each active row from this table in prioritized order according to its `snmpTlstmCertToTSN13ID` value. Each row's `snmpTlstmCertToTSN13Fingerprint` value determines whether the row is a match for the incoming connection:

- 1) If the row's `snmpTlstmCertToTSN13Fingerprint` value identifies the presented certificate, then consider the row as a successful match.
- 2) If the row's `snmpTlstmCertToTSN13Fingerprint` value identifies a locally held copy of a trusted CA certificate and that CA certificate was used to validate the path to the presented certificate, then consider the row as a successful match.

Once a matching row has been found, the `snmpTlstmCertToTSN13MapType` value can be used to determine how the `tmSecurityName` to associate with the session should be determined. See the `snmpTlstmCertToTSN13MapType` column's DESCRIPTION for details on determining the `tmSecurityName` value. If it is impossible to determine a `tmSecurityName` from the row's data combined with the data presented in the certificate, then additional rows MUST be searched looking for another potential match. If a resulting `tmSecurityName` mapped from a given row is not compatible with the needed requirements of a `tmSecurityName` (e.g., VACM imposes a 32-octet-maximum length and the certificate derived `securityName` could be longer), then it must be considered an invalid match and additional rows MUST be searched looking for another potential match.

If no matching and valid row can be found, the connection MUST be closed and SNMP messages MUST NOT be accepted over it.

Missing values of `snmpTlstmCertToTSN13ID` are acceptable and implementations should continue to the next highest numbered row. It is recommended that administrators skip index values to leave room for the insertion of future rows (for example, use values of 10 and 20 when creating initial rows).

Users are encouraged to make use of certificates with `subjectAltName` fields that can be used as `tmSecurityNames` so that a single root CA certificate can allow all child certificate's `subjectAltName` to map directly to a `tmSecurityName` via a 1:1 transformation. However, this table is flexible to allow for situations where existing deployed certificate infrastructures do not provide adequate `subjectAltName` values for use as `tmSecurityNames`.

Direct mapping from each individual certificate fingerprint to a `tmSecurityName` is possible but requires one entry in the table per `tmSecurityName` and requires more management operations to completely configure a device."

```
::= { snmpTlstmCertificateMapping 12 }
```

```

snmpTlstmCertToTSN13Entry OBJECT-TYPE
    SYNTAX      SnmpTlstmCertToTSN13Entry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A row in the snmpTlstmCertToTSN13Table that specifies a
        mapping for an incoming TLS certificate to a tmSecurityName
        to use for a connection."
    INDEX       { snmpTlstmCertToTSN13ID }
    ::= { snmpTlstmCertToTSN13Table 1 }
SnmpTlstmCertToTSN13Entry ::= SEQUENCE {
    snmpTlstmCertToTSN13ID      Unsigned32,
    snmpTlstmCertToTSN13Fingerprint  SnmpTLS13Fingerprint,
    snmpTlstmCertToTSN13MapType   AutonomousType,
    snmpTlstmCertToTSN13Data      OCTET STRING,
    snmpTlstmCertToTSN13StorageType  StorageType,
    snmpTlstmCertToTSN13RowStatus   RowStatus
}
snmpTlstmCertToTSN13ID OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A unique, prioritized index for the given entry.  Lower
        numbers indicate a higher priority."
    ::= { snmpTlstmCertToTSN13Entry 1 }
snmpTlstmCertToTSN13Fingerprint OBJECT-TYPE
    SYNTAX      SnmpTLS13Fingerprint (SIZE(2..255))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "A cryptographic hash of an X.509 certificate.  The results of
        a successful matching fingerprint to either the trusted CA in
        the certificate validation path or to the certificate itself
        is dictated by the snmpTlstmCertToTSN13MapType column."
    ::= { snmpTlstmCertToTSN13Entry 2 }
snmpTlstmCertToTSN13MapType OBJECT-TYPE
    SYNTAX      AutonomousType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Specifies the mapping type for deriving a tmSecurityName from
        a certificate.  Details for mapping of a particular type SHALL
        be specified in the DESCRIPTION clause of the OBJECT-IDENTITY
        that describes the mapping.  If a mapping succeeds it will
        return a tmSecurityName for use by the TLSTM model and
        processing stops.
        If the resulting mapped value is not compatible with the

```

needed requirements of a tmSecurityName (e.g., VACM imposes a 32-octet-maximum length and the certificate derived securityName could be longer), then future rows MUST be searched for additional snmpTlstmCertToTSN13Fingerprint matches to look for a mapping that succeeds. Suitable values for assigning to this object that are defined within the SNMP-TLS-TM-MIB can be found in the snmpTlstmCertToTSNMIdentities portion of the MIB tree."

```

DEFVAL { snmpTlstmCertSpecified }
::= { snmpTlstmCertToTSN13Entry 3 }
snmpTlstmCertToTSN13Data OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE(0..1024))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "Auxiliary data used as optional configuration information for
        a given mapping specified by the snmpTlstmCertToTSN13MapType
        column. Only some mapping systems will make use of this
        column. The value in this column MUST be ignored for any
        mapping type that does not require data present in this
        column."
    DEFVAL { "" }
    ::= { snmpTlstmCertToTSN13Entry 4 }
snmpTlstmCertToTSN13StorageType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The storage type for this conceptual row. Conceptual rows
        having the value 'permanent' need not allow write-access to
        any columnar objects in the row."
    DEFVAL      { nonVolatile }
    ::= { snmpTlstmCertToTSN13Entry 5 }
snmpTlstmCertToTSN13RowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The status of this conceptual row. This object may be used
        to create or remove rows from this table.
        To create a row in this table, an administrator must set this
        object to either createAndGo(4) or createAndWait(5).
        Until instances of all corresponding columns are appropriately
        configured, the value of the corresponding instance of the
        snmpTlstmParams13RowStatus column is notReady(3).
        In particular, a newly created row cannot be made active until
        the corresponding snmpTlstmCertToTSN13Fingerprint,
        snmpTlstmCertToTSN13MapType, and snmpTlstmCertToTSN13Data

```

columns have been set.
 The following objects may not be modified while the value of this object is active(1):

- snmpTlstmCertToTSN13Fingerprint
- snmpTlstmCertToTSN13MapType
- snmpTlstmCertToTSN13Data

An attempt to set these objects while the value of snmpTlstmParams13RowStatus is active(1) will result in an inconsistentValue error."

```
 ::= { snmpTlstmCertToTSN13Entry 6 }
snmpTlstmParams13Count OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A count of the number of entries in the
        snmpTlstmParams13Table."
 ::= { snmpTlstmCertificateMapping 13 }
snmpTlstmParams13TableLastChanged OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime.0 when the snmpTlstmParams13Table
        was last modified through any means, or 0 if it has not been
        modified since the command responder was started."
 ::= { snmpTlstmCertificateMapping 14 }
snmpTlstmParams13Table OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpTlstmParams13Entry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table is used by a TLS client when a TLS
        connection is being set up using an entry in the
        SNMP-TARGET-MIB. It extends the SNMP-TARGET-MIB's
        snmpTargetParams13Table with a fingerprint of a certificate to
        use when establishing such a TLS connection."
 ::= { snmpTlstmCertificateMapping 15 }
snmpTlstmParams13Entry OBJECT-TYPE
    SYNTAX      SnmpTlstmParams13Entry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A conceptual row containing a fingerprint hash of a locally
        held certificate for a given snmpTargetParamsEntry. The
        values in this row should be ignored if the connection that
        needs to be established, as indicated by the SNMP-TARGET-MIB
        infrastructure, is not a certificate and TLS based
```


connection. The connection SHOULD NOT be established if the certificate fingerprint stored in this entry does not point to a valid locally held certificate or if it points to an unusable certificate (such as might happen when the certificate's expiration date has been reached)."

```

INDEX      { IMPLIED snmpTargetParamsName }
::= { snmpTlstmParams13Table 1 }
snmpTlstmParams13Entry ::= SEQUENCE {
    snmpTlstmParams13ClientFingerprint SnmpTLS13Fingerprint,
    snmpTlstmParams13StorageType        StorageType,
    snmpTlstmParams13RowStatus          RowStatus
}
snmpTlstmParams13ClientFingerprint OBJECT-TYPE
SYNTAX      SnmpTLS13Fingerprint
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
    "This object stores the hash of the public portion of a
    locally held X.509 certificate. The X.509 certificate, its
    public key, and the corresponding private key will be used
    when initiating a TLS connection as a TLS client."
    ::= { snmpTlstmParams13Entry 1 }
snmpTlstmParams13StorageType OBJECT-TYPE
SYNTAX      StorageType
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
    "The storage type for this conceptual row. Conceptual rows
    having the value 'permanent' need not allow write-access to
    any columnar objects in the row."
    DEFVAL   { nonVolatile }
    ::= { snmpTlstmParams13Entry 2 }
snmpTlstmParams13RowStatus OBJECT-TYPE
SYNTAX      RowStatus
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
    "The status of this conceptual row. This object may be used
    to create or remove rows from this table.
    To create a row in this table, an administrator must set this
    object to either createAndGo(4) or createAndWait(5).
    Until instances of all corresponding columns are appropriately
    configured, the value of the corresponding instance of the
    snmpTlstmParams13RowStatus column is notReady(3).
    In particular, a newly created row cannot be made active until
    the corresponding snmpTlstmParams13ClientFingerprint column has
    been set.
    The snmpTlstmParams13ClientFingerprint object may not be
  
```

modified while the value of this object is active(1).
An attempt to set these objects while the value of
snmpTlstmParams13RowStatus is active(1) will result in
an inconsistentValue error."
 ::= { snmpTlstmParams13Entry 3 }
snmpTlstmAddr13Count OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "A count of the number of entries in the snmpTlstmAddr13Table."
 ::= { snmpTlstmCertificateMapping 16 }
snmpTlstmAddr13TableLastChanged OBJECT-TYPE
SYNTAX TimeStamp
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "The value of sysUpTime.0 when the snmpTlstmAddr13Table
 was last modified through any means, or 0 if it has not been
 modified since the command responder was started."
 ::= { snmpTlstmCertificateMapping 17 }
snmpTlstmAddr13Table OBJECT-TYPE
SYNTAX SEQUENCE OF SnmpTlstmAddr13Entry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
 "This table is used by a TLS client when a TLS
 connection is being set up using an entry in the
 SNMP-TARGET-MIB. It extends the SNMP-TARGET-MIB's
 snmpTargetAddrTable so that the client can verify that the
 correct server has been reached. This verification can use
 either a certificate fingerprint, or an identity
 authenticated via certification path validation.
 If there is an active row in this table corresponding to the
 entry in the SNMP-TARGET-MIB that was used to establish the
 connection, and the row's snmpTlstmAddr13ServerFingerprint
 column has non-empty value, then the server's presented
 certificate is compared with the
 snmpTlstmAddr13ServerFingerprint value (and the
 snmpTlstmAddr13ServerIdentity column is ignored). If the
 fingerprint matches, the verification has succeeded. If the
 fingerprint does not match, then the connection MUST be
 closed.
 If the server's presented certificate has passed
 certification path validation [RFC5280] to a configured
 trust anchor, and an active row exists with a zero-length
 snmpTlstmAddr13ServerFingerprint value, then the
 snmpTlstmAddr13ServerIdentity column contains the expected

host name. This expected host name is then compared against the server's certificate as follows:

- Implementations MUST support matching the expected host name against a `dnsName` in the `subjectAltName` extension field.
- The '*' (ASCII 0x2a) wildcard character is allowed in the `dnsName` of the `subjectAltName` extension, but only as the left-most (least significant) DNS label in that value. This wildcard matches any left-most DNS label in the server name. That is, the subject `*.example.com` matches the server names `a.example.com` and `b.example.com`, but does not match `example.com` or `a.b.example.com`. Implementations MUST support wildcards in certificates as specified above, but MAY provide a configuration option to disable them.
- If the locally configured name is an internationalized domain name, conforming implementations MUST convert it to the ASCII Compatible Encoding (ACE) format for performing comparisons, as specified in Section 7 of [RFC5280].

If the expected host name fails these conditions then the connection MUST be closed.

If there is no row in this table corresponding to the entry in the SNMP-TARGET-MIB and the server can be authorized by another, implementation-dependent means, then the connection MAY still proceed."

```
 ::= { snmpTlstmCertificateMapping 18 }
snmpTlstmAddr13Entry OBJECT-TYPE
SYNTAX      SnmpTlstmAddr13Entry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "A conceptual row containing a copy of a certificate's
    fingerprint for a given snmpTargetAddrEntry. The values in
    this row should be ignored if the connection that needs to be
    established, as indicated by the SNMP-TARGET-MIB
    infrastructure, is not a TLS based connection. If an
    snmpTlstmAddr13Entry exists for a given snmpTargetAddrEntry,
    then the presented server certificate MUST match or the
    connection MUST NOT be established. If a row in this table
    does not exist to match an snmpTargetAddrEntry row, then the
    connection SHOULD still proceed if some other certificate
    validation path algorithm (e.g., RFC 5280) can be used."
INDEX       { IMPLIED snmpTargetAddrName }
 ::= { snmpTlstmAddr13Table 1 }
SnmpTlstmAddr13Entry ::= SEQUENCE {
    snmpTlstmAddr13ServerFingerprint    SnmpTLS13Fingerprint,
    snmpTlstmAddr13ServerIdentity        SnmpAdminString,
    snmpTlstmAddr13StorageType           StorageType,
```

```

        snmpTlstmAddr13RowStatus          RowStatus
    }
snmpTlstmAddr13ServerFingerprint OBJECT-TYPE
    SYNTAX          SnmpTLS13Fingerprint
    MAX-ACCESS      read-create
    STATUS           current
    DESCRIPTION
        "A cryptographic hash of a public X.509 certificate. This
        object should store the hash of the public X.509 certificate
        that the remote server should present during the TLS
        connection setup. The fingerprint of the presented
        certificate and this hash value MUST match exactly or the
        connection MUST NOT be established."
    DEFVAL { "" }
    ::= { snmpTlstmAddr13Entry 1 }
snmpTlstmAddr13ServerIdentity OBJECT-TYPE
    SYNTAX          SnmpAdminString
    MAX-ACCESS      read-create
    STATUS           current
    DESCRIPTION
        "The reference identity to check against the identity
        presented by the remote system."
    DEFVAL { "" }
    ::= { snmpTlstmAddr13Entry 2 }
snmpTlstmAddr13StorageType OBJECT-TYPE
    SYNTAX          StorageType
    MAX-ACCESS      read-create
    STATUS           current
    DESCRIPTION
        "The storage type for this conceptual row. Conceptual rows
        having the value 'permanent' need not allow write-access to
        any columnar objects in the row."
    DEFVAL          { nonVolatile }
    ::= { snmpTlstmAddr13Entry 3 }
snmpTlstmAddr13RowStatus OBJECT-TYPE
    SYNTAX          RowStatus
    MAX-ACCESS      read-create
    STATUS           current
    DESCRIPTION
        "The status of this conceptual row. This object may be used
        to create or remove rows from this table.
        To create a row in this table, an administrator must set this
        object to either createAndGo(4) or createAndWait(5).
        Until instances of all corresponding columns are
        appropriately configured, the value of the
        corresponding instance of the snmpTlstmAddr13RowStatus
        column is notReady(3).
        In particular, a newly created row cannot be made active until

```

the corresponding snmpTlstmAddr13ServerFingerprint column has been set.

Rows MUST NOT be active if the snmpTlstmAddr13ServerFingerprint column is blank and the snmpTlstmAddr13ServerIdentity is set to '*' since this would insecurely accept any presented certificate.

The snmpTlstmAddr13ServerFingerprint object may not be modified while the value of this object is active(1).

An attempt to set these objects while the value of snmpTlstmAddr13RowStatus is active(1) will result in an inconsistentValue error."

```

 ::= { snmpTlstmAddr13Entry 4 }
-- *****
-- snmpTlstmNotifications - Notifications Information
-- *****
snmpTlstmServerCertificateUnknown NOTIFICATION-TYPE
  OBJECTS { snmpTlstmSessionUnknownServerCertificate }
  STATUS current
  DESCRIPTION
    "Notification that the server certificate presented by an SNMP
    over (D)TLS server was invalid because no configured
    fingerprint or CA was acceptable to validate it. This may be
    because there was no entry in the snmpTlstmAddrTable (or
    snmpTlstmAddr13Table) or
    because no path could be found to known Certification
    Authority.
    To avoid notification loops, this notification MUST NOT be
    sent to servers that themselves have triggered the
    notification."
  ::= { snmpTlstmNotifications 1 }
snmpTlstmServerInvalidCertificate NOTIFICATION-TYPE
  OBJECTS { snmpTlstmAddrServerFingerprint,
            snmpTlstmSessionInvalidServerCertificates}
  STATUS deprecated
  DESCRIPTION
    "Notification that the server certificate presented by an SNMP
    over (D)TLS server could not be validated even if the
    fingerprint or expected validation path was known. That is, a
    cryptographic validation error occurred during certificate
    validation processing.
    To avoid notification loops, this notification MUST NOT be
    sent to servers that themselves have triggered the
    notification."
  ::= { snmpTlstmNotifications 2 }
snmpTlstmServerInvalidCertificate13 NOTIFICATION-TYPE
  OBJECTS { snmpTlstmAddr13ServerFingerprint,
            snmpTlstmSessionInvalidServerCertificates}
  STATUS current

```

DESCRIPTION

"Notification that the server certificate presented by an SNMP over TLS server could not be validated even if the fingerprint or expected validation path was known. That is, a cryptographic validation error occurred during certificate validation processing.

To avoid notification loops, this notification MUST NOT be sent to servers that themselves have triggered the notification."

```
 ::= { snmpTlstmNotifications 3 }
-- *****
-- snmpTlstmCompliances - Conformance Information
-- *****
snmpTlstmCompliances OBJECT IDENTIFIER ::= { snmpTlstmConformance 1 }
snmpTlstmGroups OBJECT IDENTIFIER ::= { snmpTlstmConformance 2 }
-- *****
-- Compliance statements
-- *****
snmpTlstmCompliance MODULE-COMPLIANCE
  STATUS      deprecated
  DESCRIPTION
    "The compliance statement for SNMP engines that support the
    SNMP-TLS-TM-MIB"
  MODULE
    MANDATORY-GROUPS { snmpTlstmStatsGroup,
                        snmpTlstmIncomingGroup,
                        snmpTlstmOutgoingGroup,
                        snmpTlstmNotificationGroup }
  ::= { snmpTlstmCompliances 1 }
snmpTlstmCompliance13 MODULE-COMPLIANCE
  STATUS      current
  DESCRIPTION
    "The compliance statement for SNMP engines that support the
    SNMP-TLS-TM-MIB"
  MODULE
    MANDATORY-GROUPS { snmpTlstmStatsGroup,
                        snmpTlstmIncoming13Group,
                        snmpTlstmOutgoing13Group,
                        snmpTlstmNotification13Group }
  ::= { snmpTlstmCompliances 2 }
-- *****
-- Units of conformance
-- *****
snmpTlstmStatsGroup OBJECT-GROUP
  OBJECTS {
    snmpTlstmSessionOpens,
    snmpTlstmSessionClientCloses,
    snmpTlstmSessionOpenErrors,
```

```

        snmpTlstmSessionAccepts,
        snmpTlstmSessionServerCloses,
        snmpTlstmSessionNoSessions,
        snmpTlstmSessionInvalidClientCertificates,
        snmpTlstmSessionUnknownServerCertificate,
        snmpTlstmSessionInvalidServerCertificates,
        snmpTlstmSessionInvalidCaches
    }
    STATUS          current
    DESCRIPTION
        "A collection of objects for maintaining
        statistical information of an SNMP engine that
        implements the SNMP TLS Transport Model."
    ::= { snmpTlstmGroups 1 }
snmpTlstmIncomingGroup OBJECT-GROUP
    OBJECTS {
        snmpTlstmCertToTSNCount,
        snmpTlstmCertToTSNTableLastChanged,
        snmpTlstmCertToTSNFingerprint,
        snmpTlstmCertToTSNMapType,
        snmpTlstmCertToTSNData,
        snmpTlstmCertToTSNStorageType,
        snmpTlstmCertToTSNRowStatus
    }
    STATUS          deprecated
    DESCRIPTION
        "A collection of objects for maintaining
        incoming connection certificate mappings to
        tmSecurityNames of an SNMP engine that implements the
        SNMP TLS Transport Model."
    ::= { snmpTlstmGroups 2 }
snmpTlstmOutgoingGroup OBJECT-GROUP
    OBJECTS {
        snmpTlstmParamsCount,
        snmpTlstmParamsTableLastChanged,
        snmpTlstmParamsClientFingerprint,
        snmpTlstmParamsStorageType,
        snmpTlstmParamsRowStatus,
        snmpTlstmAddrCount,
        snmpTlstmAddrTableLastChanged,
        snmpTlstmAddrServerFingerprint,
        snmpTlstmAddrServerIdentity,
        snmpTlstmAddrStorageType,
        snmpTlstmAddrRowStatus
    }
    STATUS          deprecated
    DESCRIPTION
        "A collection of objects for maintaining

```

```
        outgoing connection certificates to use when opening
        connections as a result of SNMP-TARGET-MIB settings."
 ::= { snmpTlstmGroups 3 }
snmpTlstmNotificationGroup NOTIFICATION-GROUP
NOTIFICATIONS {
    snmpTlstmServerCertificateUnknown,
    snmpTlstmServerInvalidCertificate
}
STATUS deprecated
DESCRIPTION
    "Notifications"
 ::= { snmpTlstmGroups 4 }
snmpTlstmIncomingI3Group OBJECT-GROUP
OBJECTS {
    snmpTlstmCertToTSN13Count,
    snmpTlstmCertToTSN13TableLastChanged,
    snmpTlstmCertToTSN13Fingerprint,
    snmpTlstmCertToTSN13MapType,
    snmpTlstmCertToTSN13Data,
    snmpTlstmCertToTSN13StorageType,
    snmpTlstmCertToTSN13RowStatus
}
STATUS current
DESCRIPTION
    "A collection of objects for maintaining
    incoming connection certificate mappings to
    tmSecurityNames of an SNMP engine that implements the
    SNMP TLS 1.3 Transport Model."
 ::= { snmpTlstmGroups 5 }
snmpTlstmOutgoingI3Group OBJECT-GROUP
OBJECTS {
    snmpTlstmParamsI3Count,
    snmpTlstmParamsI3TableLastChanged,
    snmpTlstmParamsI3ClientFingerprint,
    snmpTlstmParamsI3StorageType,
    snmpTlstmParamsI3RowStatus,
    snmpTlstmAddrI3Count,
    snmpTlstmAddrI3TableLastChanged,
    snmpTlstmAddrI3ServerFingerprint,
    snmpTlstmAddrI3ServerIdentity,
    snmpTlstmAddrI3StorageType,
    snmpTlstmAddrI3RowStatus
}
STATUS current
DESCRIPTION
    "A collection of objects for maintaining
    outgoing connection certificates to use when opening
    TLS 1.3 connections as a result of SNMP-TARGET-MIB settings."
```



```
 ::= { snmpTlstmGroups 6 }
snmpTlstmNotification13Group NOTIFICATION-GROUP
NOTIFICATIONS {
    snmpTlstmServerCertificateUnknown,
    snmpTlstmServerInvalidCertificate13
}
STATUS current
DESCRIPTION
    "Notifications for the SNMP TLS 1.3 Transport Model"
 ::= { snmpTlstmGroups 7 }
END
```

5. Security Considerations

This document updates a transport model that permits SNMP to utilize TLS security services. The security threats and how the TLS transport model mitigates these threats are covered throughout this document and in [RFC6353]. Security considerations for TLS are described in Section 10 and Appendix E of TLS 1.3 [RFC8446].

5.1. MIB Module Security

There are a number of management objects defined in this MIB module with a MAX-ACCESS clause of read-write and/or read-create. Such objects might be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations. These are the tables and objects and their sensitivity/vulnerability:

- * The `snmpTlstmParams13Table` can be used to change the outgoing X.509 certificate used to establish a TLS connection. Modifications to objects in this table need to be adequately authenticated since modifying the values in this table will have profound impacts to the security of outbound connections from the device. Since knowledge of authorization rules and certificate usage mechanisms might be considered sensitive, protection from disclosure of the SNMP traffic via encryption is automatically achieved via TLS 1.3.
- * The `snmpTlstmAddr13Table` can be used to change the expectations of the certificates presented by a remote TLS server. Modifications to objects in this table need to be adequately authenticated since modifying the values in this table will have profound impacts to the security of outbound connections from the device. Since knowledge of authorization rules and certificate usage mechanisms might be considered sensitive, protection from disclosure of the SNMP traffic via encryption is automatically achieved via TLS 1.3.

- * The `snmpTlstmCertToTSN13Table` is used to specify the mapping of incoming X.509 certificates to `tmSecurityNames`, which eventually get mapped to an SNMPv3 `securityName`. Modifications to objects in this table need to be adequately authenticated since modifying the values in this table will have profound impacts to the security of incoming connections to the device. Since knowledge of authorization rules and certificate usage mechanisms might be considered sensitive, protection from disclosure of the SNMP traffic via encryption is automatically achieved via TLS 1.3. When this table contains a significant number of rows it might affect the system performance when accepting new TLS connections.

Some of the readable objects in this MIB module (i.e., objects with a MAX-ACCESS other than not-accessible) might be considered sensitive or vulnerable in some network environments. It is thus important to control even GET and/or NOTIFY access to these objects and encrypt the values of these objects when sending them over the network via SNMP. These are the tables and objects and their sensitivity/vulnerability:

- * This MIB contains a collection of counters that monitor the TLS connections being established with a device. Since knowledge of connection and certificate usage mechanisms might be considered sensitive, protection from disclosure of the SNMP traffic via encryption is automatically achieved via TLS 1.3.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example, by using IPsec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.

As defined in Section 2.4, TLSTM clients and servers MUST NOT request, offer, or use SNMPv1 or SNMPv2c message processing described in [RFC3584], or the User-based Security Model of SNMPv3. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

6. IANA Considerations

This document has no IANA actions beyond those performed as a part of [RFC6353].

7. Acknowledgements

Acknowledgements This document is based on [RFC6353]. This document was reviewed by the following people who helped provide useful comments: Michaela Vanderveen.

8. References

8.1. Normative References

- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021,
<<https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3584] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", BCP 74, RFC 3584, DOI 10.17487/RFC3584, August 2003,
<<https://www.rfc-editor.org/info/rfc3584>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008,
<<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5591] Harrington, D. and W. Hardaker, "Transport Security Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 5591, DOI 10.17487/RFC5591, June 2009,
<<https://www.rfc-editor.org/info/rfc5591>>.
- [RFC6353] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 6353, DOI 10.17487/RFC6353, July 2011,
<<https://www.rfc-editor.org/info/rfc6353>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,
<<https://www.rfc-editor.org/info/rfc8446>>.

- [STD62] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.
- Case, J., Harrington, D., Presuhn, R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3412, December 2002.
- Levi, D., Meyer, P., and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.
- Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.
- Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.
- Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.
- Presuhn, R., Ed., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.
- Presuhn, R., Ed., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

8.2. Informative References

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [STD58] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIV2", STD 58, RFC 2579, April 1999.

McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Conformance Statements for SMIV2", STD 58, RFC 2580, April 1999.

Appendix A. Target and Notification Configuration Example

The following sections describe example configuration for the SNMP-TLS-TM-MIB, the SNMP-TARGET-MIB, the NOTIFICATION-MIB, and the SNMP-VIEW-BASED-ACM-MIB.

A.1. Configuring a Notification Originator

The following row adds the "Joe Cool" user to the "administrators" group:

```
vacmSecurityModel          = 4 (TSM)
vacmSecurityName           = "Joe Cool"
vacmGroupName              = "administrators"
vacmSecurityToGroupStorageType = 3 (nonVolatile)
vacmSecurityToGroupStatus   = 4 (createAndGo)
```

The following row configures the snmpTlstmAddr13Table to use certificate path validation and to require the remote notification receiver to present a certificate for the "server.example.org" identity.

```
snmpTargetAddrName        = "toNRAddr"
snmpTlstmAddr13ServerFingerprint = ""
snmpTlstmAddr13ServerIdentity = "server.example.org"
snmpTlstmAddr13StorageType = 3 (nonVolatile)
snmpTlstmAddr13RowStatus   = 4 (createAndGo)
```

The following row configures the snmpTargetAddrTable to send notifications using TLS/TCP to the snmpTls-trap port at 192.0.2.1:

```
snmpTargetAddrName        = "toNRAddr"
snmpTargetAddrTDomain     = snmpTLSTCPDomain
snmpTargetAddrTAddress     = "192.0.2.1:10162"
snmpTargetAddrTimeout     = 1500
snmpTargetAddrRetryCount  = 3
snmpTargetAddrTagList     = "toNRTag"
snmpTargetAddrParams      = "toNR" (MUST match below)
snmpTargetAddrStorageType = 3 (nonVolatile)
snmpTargetAddrRowStatus   = 4 (createAndGo)
```

The following row configures the `snmpTargetParamsTable` to send the notifications to "Joe Cool", using `authPriv` SNMPv3 notifications through the `TransportSecurityModel` [[RFC5591]]:

```

snmpTargetParamsName      = "toNR"      (MUST match above)
snmpTargetParamsMPModel   = 3 (SNMPv3)
snmpTargetParamsSecurityModel = 4 (TransportSecurityModel)
snmpTargetParamsSecurityName = "Joe Cool"
snmpTargetParamsSecurityLevel = 3          (authPriv)
snmpTargetParamsStorageType = 3          (nonVolatile)
snmpTargetParamsRowStatus  = 4          (createAndGo)

```

A.2. Configuring TLSTM to Utilize a Simple Derivation of `tmSecurityName`

The following row configures the `snmpTlstmCertToTSN13Table` to map a validated client certificate, referenced by the client's public X.509 hash fingerprint, to a `tmSecurityName` using the `subjectAltName` component of the certificate.

```

snmpTlstmCertToTSN13ID      = 1
                           (chosen by ordering preference)
snmpTlstmCertToTSN13Fingerprint = HASH (appropriate fingerprint)
snmpTlstmCertToTSN13MapType  = snmpTlstmCertSANAny
snmpTlstmCertToTSN13Data     = "" (not used)
snmpTlstmCertToTSN13StorageType = 3 (nonVolatile)
snmpTlstmCertToTSN13RowStatus  = 4 (createAndGo)

```

This type of configuration should only be used when the naming conventions of the (possibly multiple) Certification Authorities are well understood, so two different principals cannot inadvertently be identified by the same derived `tmSecurityName`.

A.3. Configuring TLSTM to Utilize Table-Driven Certificate Mapping

The following row configures the `snmpTlstmCertToTSN13Table` to map a validated client certificate, referenced by the client's public X.509 hash fingerprint, to the directly specified `tmSecurityName` of "Joe Cool".

```

snmpTlstmCertToTSN13ID      = 2
                           (chosen by ordering preference)
snmpTlstmCertToTSN13Fingerprint = HASH (appropriate fingerprint)
snmpTlstmCertToTSN13MapType  = snmpTlstmCertSpecified
snmpTlstmCertToTSN13SecurityName = "Joe Cool"
snmpTlstmCertToTSN13StorageType = 3 (nonVolatile)
snmpTlstmCertToTSN13RowStatus  = 4 (createAndGo)

```

Author's Address

Kenneth Vaughn (editor)
Trevilon LLC
6606 FM 1488 RD
Suite 148-503
Magnolia, TX 77354
United States of America

Phone: +1 571 331 5670
Email: kvaughn@trevilon.com