Dynamic Host Configuration
Internet-Draft
Intended status: Standards Track
Expires: 14 October 2024

W. Kumari
Google, LLC
S. Krishnan
Cisco Systems, Inc.
R. Asati
Independent
L. Colitti
J. Linkova
Google, LLC
S. Jiang
Beijing University of Posts and Telecommunications

12 April 2024

Registering Self-generated IPv6 Addresses using DHCPv6
draft-ietf-dhc-addr-notification-11

Abstract

This document defines a method to inform a DHCPv6 server that a device has one or more self-generated or statically configured addresses.

About This Document

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

The latest revision of this draft can be found at https://wkumari.github.io/draft-wkumari-dhc-addr-notification/draft-wkumari-dhc-addr-notification.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-dhc-addr-notification/.

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

Source for this draft and an issue tracker can be found at https://github.com/wkumari/draft-wkumari-dhc-addr-notification.

Status of This Memo

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

This Internet-Draft will expire on 14 October 2024.

Table of Contents

1. Introduction

It is very common operational practice, especially in enterprise networks, to use IPv4 DHCP logs for troubleshooting or forensics purposes. Examples of this include a help desk dealing with a ticket such as "The CEO's laptop cannot connect to the printer"; if the MAC address of the printer is known (for example from an inventory system), the printer's IPv4 address can be retrieved from the DHCP log or lease table and the printer pinged to determine if it is reachable. Another common example is a Security Operations team discovering suspicious events in outbound firewall logs and then consulting DHCP logs to determine which employee's laptop had that IPv4 address at that time so that they can quarantine it and remove the malware.

This operational practice relies on the DHCP server knowing the IP address assignments. Therefore, the practice does not work if static IP addresses are manually configured on devices or self-assigned addresses (such as when self-configuring an IPv6 address using SLAAC [RFC4862]) are used.

The lack of this parity with IPv4 is one of the reasons that may be hindering IPv6 deployment, especially in enterprise networks.

This document provides a mechanism for a device to inform the DHCPv6 server that the device has a self-configured IPv6 address (or has a statically configured address), and thus provides parity with IPv4 in this aspect.

2. Conventions and Definitions

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

3. Registration Mechanism Overview

The DHCPv6 protocol is used as the address registration protocol when a DHCPv6 server performs the role of an address registration server. This document introduces a new Address Registration (OPTION_ADDR_REG_ENABLE) option which indicates that the server supports the registration mechanism. Before registering any addresses, the client MUST determine whether the network supports address registration. It can do this by including the Address Registration option code the Option Request option (see Section 21.7 of [RFC8415]) of the Information-Request, Solicit, Request, Renew, or

Rebind messages it sends to the server as part of the regular stateless or stateful DHCPv6 configuration process. If the server supports address registration, it includes an Address Registration option in its Reply message. If the network does not support (or is not willing to receive) any address registration information, the client MUST NOT register any addresses. Otherwise, the client registers addresses as described below.

After successfully assigning a self-generated IPv6 address on one of its interfaces, a client implementing this specification SHOULD multicast an ADDR-REG-INFORM message in order to inform the DHCPv6 server that this self-generated address is in use. Each ADDR-REG-INFORM message contains an DHCPv6 IA Address option [RFC8415] to specify the address to being registered.

The address registration mechanism overview is shown in Fig.1.

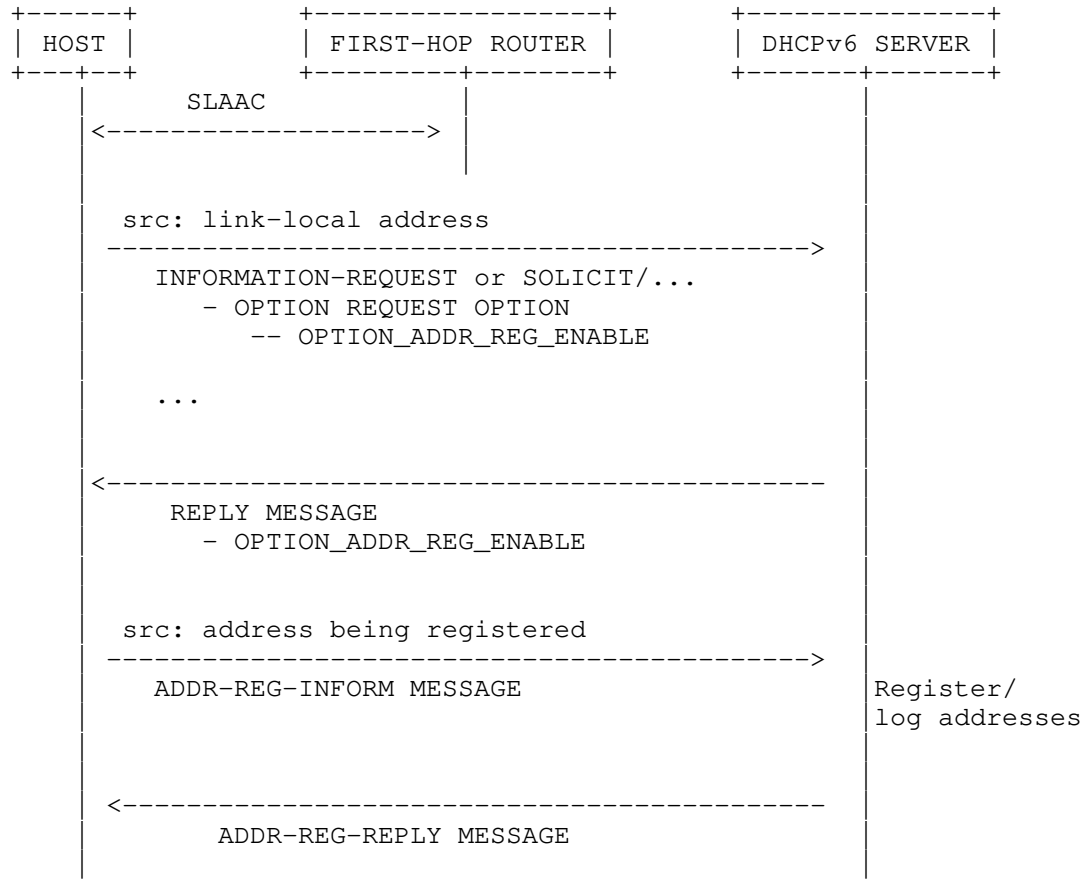
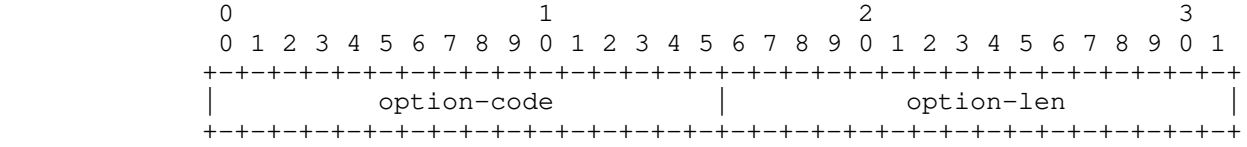


Figure 1: Address Registration Procedure Overview

4. DHCPv6 Address Registration Procedure

4.1. DHCPv6 Address Registration Option

The DHCPv6 server includes an Address Registration option (OPTION_ADDR_REG_ENABLE) to indicate that the server supports the mechanism described in this document. The format of the Address Registration option is described as follows:



option-code OPTION_ADDR_REG_ENABLE (TBA0)

option-len 0

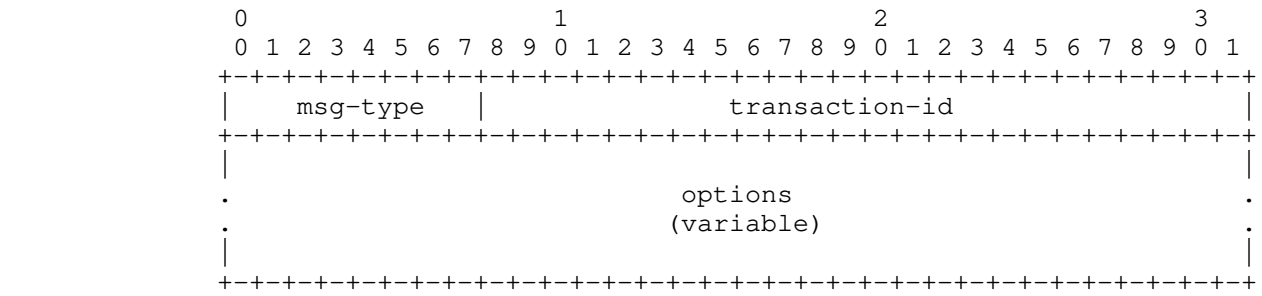
Figure 2: DHCPv6 Address Registration option

If a client has the address registration mechanism enabled, it SHOULD include this option in all Option Request options that it sends.

A server which is configured to support the address registration mechanism MUST include this option in Reply messages.

4.2. DHCPv6 Address Registration Request Message

The DHCPv6 client sends an ADDR-REG-INFORM message to inform that an IPv6 address is in use. The format of the ADDR-REG-INFORM message is described as follows:



msg-type — Identifies the DHCPv6 message type; Set to ADDR-REG-INFORM (TBA1).

transaction-id — The transaction ID for this message exchange.

options — Options carried in this message.

Figure 3: DHCPv6 ADDR-REG-INFORM message

The client MUST generate a transaction ID as described in [RFC8415] and insert this value in the "transaction-id" field.

The client MUST include the Client Identifier option [RFC8415] in the ADDR-REG-INFORM message.

The ADDR-REG-INFORM message MUST NOT contain the Server Identifier option and MUST contain exactly one IA Address option containing the address being registered. The valid-lifetime and preferred-lifetime fields in the option MUST match the current Valid Lifetime and Preferred Lifetime of the address being registered.

The ADDR-REG-INFORM message is dedicated for clients to initiate an address registration request toward an address registration server. Consequently, clients MUST NOT put any Option Request Option(s) in the ADDR-REG-INFORM message. Clients MAY include other options, such as the Client FQDN Option [RFC4704].

The client sends the DHCPv6 ADDR-REG-INFORM message to the All_DHCP_Relay_Agents_and_Servers multicast address (ff02::1:2). The client MUST send separate messages for each address being registered.

Unlike other types of messages, which are sent from the link-local address of the client, the ADDR-REG-INFORM message MUST be sent from the address being registered. This is primarily for "fate sharing" purposes - for example, if the network implements some form of layer-2 security to prevent a client from spoofing other clients' MAC addresses, this prevents an attacker from spoofing ADDR-REG-INFORM messages.

On clients with multiple interfaces, the client MUST only send the packet on the network interface that has the address being registered, even if it has multiple interfaces with different addresses. If the same address is configured on multiple interfaces, then the client MUST send ADDR-REG-INFORM each time the address is configured on an interface that did not previously have it, and refresh each registration independently from the others.

The client MUST only send the ADDR-REG-INFORM message for valid ([RFC4862]) addresses of global scope ([RFC4007]). This includes ULA addresses, which are defined in [RFC4193] to have global scope. The client MUST NOT send the ADDR-REG-INFORM message for addresses configured by DHCPv6.

The client SHOULD NOT send the ADDR-REG-INFORM message if it has not received any Router Advertisement message with either M or O flags set to 1.

Clients MUST discard any received ADDR-REG-INFORM messages.

4.2.1. Server message processing

Servers MUST discard any ADDR-REG-INFORM messages that meet any of the following conditions:

* the message does not include a Client Identifier option;

* the message includes a Server Identifier option;

* the message does not include the IA Address option, or the IP address in the IA Address option does not match the source address of the original ADDR-REG-INFORM message sent by the client. The source address of the original message is the source IP address of the packet if it is not relayed, or the Peer-Address field of the innermost Relay-Forward message if it is relayed.

* the message includes an Option Request Option.

If the message is not discarded, the address registration server SHOULD verify that the address being registered is "appropriate to the link" as defined by [RFC8415] or within a prefix delegated to the client. Otherwise, it MUST drop the message, and SHOULD log this fact. Otherwise, the server:

* SHOULD register or update a binding between the provided Client Identifier and IPv6 address in its database. The lifetime of the binding is equal to the Valid Lifetime of the address reported by the client. If there is already a binding between the registered address and another client, the server SHOULD log the fact and update the binding.

* SHOULD log the address registration information (as is done normally for clients to which it has assigned an address), unless configured not to do so. The server SHOULD log the client DUID and the link-layer address, if available. The server MAY log any other information.

* SHOULD mark the address as unavailable for use and not include it in future ADVERTISE messages.

* MUST send back an ADDR-REG-REPLY message to ensure the client does not retransmit.

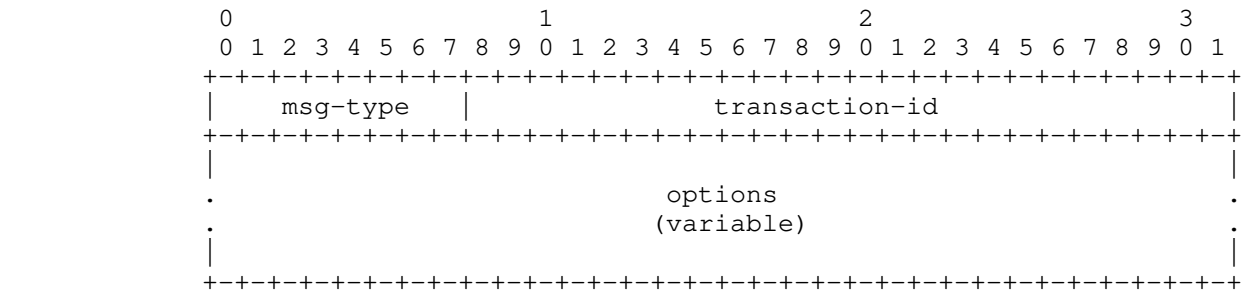
If a client is multihomed (connected to multiple administrative domains, each operating its own DHCPv6 infrastructure), the requirement to verify that the registered address is appropriate for the link or belongs to a delegated prefix ensures that each DHCPv6 server only registers bindings for addresses from the given administrative domain.

Although a client "MUST NOT send the ADDR-REG-INFORM message for addresses configured by DHCPv6", if a server does receive such a message, it should log and discard it.

DHCPv6 relay agents and switches that relay address registration messages directly from clients MUST include the client's link-layer address in the relayed message using the Client Link-Layer Address option ([RFC6939]) if they would do so for other DHCPv6 client messages such as SOLICIT, REQUEST, and REBIND.

4.3. DHCPv6 Address Registration Acknowledgement

The server MUST acknowledge receipt of a valid ADDR-REG-INFORM message by sending back an ADDR-REG-REPLY message. The format of the ADDR-REG-REPLY message is described as follows:



msg-type — Identifies the DHCPv6 message type; Set to ADDR-REG-REPLY (TBA2).

transaction-id — The transaction ID for this message exchange.

options — Options carried in this message.

Figure 4: DHCPv6 ADDR-REG-REPLY message

If the ADDR-REG-INFORM message that the server is replying to was not relayed, then the IPv6 destination address of the message MUST be the address being registered. If the ADDR-REG-INFORM message was relayed, then the server MUST construct the Relay-reply message as specified in [RFC8415] section 19.3.

The server MUST copy the transaction-id from the ADDR-REG-INFORM message to the transaction-id field of the ADDR-REG-REPLY.

The ADDR-REG-REPLY message MUST contain an IA Address option for the address being registered. The option MUST be identical to the one in the ADDR-REG-INFORM message that the server is replying to.

Servers MUST ignore any received ADDR-REG-REPLY messages.

Clients MUST discard any ADDR-REG-REPLY messages that meet any of the following conditions:

* The IPv6 destination address does not match the address being registered.

* The IA-Address option does not match the address being registered.

* The address being registered is not assigned to the interface receiving the message.

* The transaction-id does not match the transaction-id the client used in the corresponding ADDR-REG-INFORM message.

The ADDR-REG-REPLY message only indicates that the ADDR-REG-INFORM message has been received and that the client should not retransmit it. The ADDR-REG-REPLY message MUST NOT be considered as any indication of the address validity and MUST NOT be required for the address to be usable. DHCPv6 relays, or other devices that snoop ADDR-REG-REPLY messages, MUST NOT add or alter any forwarding or security state based on the ADDR-REG-REPLY message.

4.4. Signalling Address Registration Support

The client MUST NOT register addresses using this mechanism unless the network's DHCPv6 servers support address registration. The client can discover this using the OPTION_ADDR_REG_ENABLE option. The client SHOULD include this option code in all Option Request options that it sends. If the client receives and processes a Reply message with the OPTION_ADDR_REG_ENABLE option, it concludes that the network supports address registration. When the client detects that the network supports address registration, it SHOULD start the registration process and immediately register any addresses that are already in use. The client SHOULD NOT stop registering addresses until it disconnects from the link, even if subsequent Reply or Advertise messages do not contain the OPTION_ADDR_REG_ENABLE option.

The client MUST discover whether the network supports address registration every time it connects to a network or when it detects it has moved to a new link, without utilizing any prior knowledge about address registration support by that network or link. This host behavior allows networks to progressively roll out support for the address registration option across the DHCPv6 infrastructure without causing clients to frequently stop and re-start address registration if some of the network's DHCPv6 servers support it and some of them do not.

4.5. Retransmission

To reduce the effects of packet loss on registration, the client SHOULD retransmit the registration message. Retransmissions SHOULD follow the standard retransmission logic specified by section 15 of [RFC8415] with the following default parameters:

* IRT 1 sec

* MRC 3

The client SHOULD allow these parameters to be configured by the administrator.

To comply with section 16.1 of [RFC8415], the client MUST leave the transaction ID unchanged in retransmissions of an ADDR-REG-INFORM message. When the client retranmits the registration message, the lifetimes in the packet MUST be updated so that they match the current lifetimes of the address.

If an ADDR-REG-REPLY message is received for the address being registered, the client MUST stop retransmission.

4.6. Registration Expiry and Refresh

The client MUST refresh registrations to ensure that the server is always aware of which addresses are still valid. The client SHOULD perform refreshes as described below.

We define a function AddrRegRefreshInterval(address) as min(4 hours, 80% of the address's current Valid Lifetime). When calculating this value, the client applies a multiplier of AddrRegDesyncMultiplier to avoid synchronization causing a large number of registration messages from different clients at the same time. AddrRegDesyncMultiplier is between 0.9 and 1.1 and is chosen by the client when it starts the registration process, to ensure that refreshes for addresses with the same lifetime are coalesced (see below).

Whenever the client registers or refreshes an address, it calculates a NextAddrRegRefreshTime for that address as AddrRegRefreshInterval seconds in the future, but does not schedule any refreshes.

Whenever the client receives a Prefix Information option [RFC4861] which changes the Valid Lifetime of an existing address by more than 1%, then the client calculates a new AddrRegRefreshInterval. The client schedules a refresh for min(now + AddrRegRefreshInterval, NextAddrRegRefreshTime). If the refresh would be scheduled in the past, then the refresh occurs immediately.

When a refresh is performed, the client MAY refresh all addresses assigned to the interface that are scheduled to be refreshed within the next AddrRegRefreshCoalesce seconds. The value of AddrRegRefreshCoalesce is implementation-dependent, and a suggested default is 60 seconds.

Justification: this algorithm ensures that refreshes are not sent too frequently, while ensuring that the server never believes that the address has expired when it has not. Specifically, after every registration:

* If the client never receives a PIO that changes the lifetime (e.g., if no further PIOs are received, or if all PIO lifetimes decrease in step with the passage of time), then no refreshes occur. Refreshes are not necessary, because the address expires at the time the server expects it to expire.

* Any time a PIO changes the lifetime of the address (i.e., changes the time at which the address will expire) the client ensures that a refresh is scheduled, so that server will be informed of the new expiry.

* Because AddrRegDesyncMultiplier is at most 1.1, the refresh never occurs later than a point 88% between the time when the address was registered and the time when the address will expire. This allows the client to retransmit the registration for up to 12% of the original interval before it expires. This may not be possible if the network sends an RA very close to the time when the address would have expired. In this case, the client refreshes immediately, which is the best it can do.

* The 1% tolerance ensures that the client will not refresh or reschedule refreshes if the Valid Lifetime experiences minor changes due to transmission delays or clock skew between the client and the router(s) sending the Router Advertisement.

* AddrRegRefreshCoalesce allows battery-powered hosts to wake up less often. In particular, it allows the client to coalesce refreshes for multiple addresses formed from the same prefix, such as the stable and privacy addresses. Higher values will result in fewer wakeups, but may result in more network traffic, because if a refresh is sent early, then the next RA received will cause the client to immediately send a refresh message.

* In typical networks, the lifetimes in periodic Router Advertisements either contain constant values, or values that decrease over time to match the another lifetime, such as the lifetime of a prefix delegated to the network. In both these cases, this algorithm will refresh order of once per address lifetime, which is similar to the number of refreshes that are necessary using stateful DHCPv6.

Registration refresh packets SHOULD be retransmitted using the same logic as described in the 'Retransmission' section above.

The client MUST generate a new transaction ID when refreshing the registration.

When the Client-Identifier-to-IPv6-address binding has expired, the server SHOULD remove it and consider the address as available for use.

The client MAY choose to notify the server when an address is no longer being used (e.g., if the client is disconnecting from the network, the address lifetime expired, or the address is being removed from the interface). To indicate that the address is not being used anymore the client MUST set the preferred-lifetime and valid-lifetime fields of the IA Address option to zero. If the server receives a message with a valid-lifetime of zero, it SHOULD act as if the address has expired.

5. Host configuration

DHCP clients SHOULD allow the administrator to disable sending ADDR-REG-INFORM messages. This could be used, for example, to reduce network traffic on networks where the servers are known not to support the message type. Sending the messages SHOULD be enabled by default.

6. Security Considerations

An attacker may attempt to register a large number of addresses in quick succession in order to overwhelm the address registration server and / or fill up log files. Similar attack vectors exist today, e.g. an attacker can DoS the server with messages contained spoofed DHCP Unique Identifiers (DUIDs) [RFC8415].

If a network is using FCFS SAVI [RFC6620], then the DHCPv6 server can trust that the ADDR-REG-INFORM message was sent by the legitimate holder of the address. This prevents a host from registering an address owned by another host.

If the network doesn't have MLD snooping enabled, then IPv6 link-local multicast traffic is effectively transmitted as broadcast. In such networks, an on-link attacker listening to DHCPv6 messages might obtain information about IPv6 addresses assigned to the host. However, hiding information about the specific IPv6 address should not be considered a security measure, as such information is usually disclosed via Duplicate Address Detection [RFC4862] to all nodes anyway if MLD snooping is not enabled.

If MLD snooping is enabled, an attacker might be able to join the All_DHCP_Relay_Agents_and_Servers multicast address (ff02::1:2) group to listen for address registration messages. However the same result can be achieved by joining the All Routers Address (ff02::2) group and listen to Gratuitous Neighbor Advertisement messages [RFC9131].

It should be noted that this particular scenario shares the fate with DHCPv6 address assignment: if an attacker can join the All_DHCP_Relay_Agents_and_Servers multicast group, they would be able to monitor all DHCPv6 messages sent from the client to DHCPv6 servers and relays, and therefore obtain the information about addresses being assigned via DHCPv6. Layer-2 isolation allows to mitigate this threat by blocking onlink peer-to-peer communication between hosts.

One of the use cases for the mechanism described in this document is to identify sources of malicious traffic after the fact. Note, however, that as the device itself is responsible for informing the DHCPv6 server that it is using an address, a malicious or compromised device can simply not send the ADDR-REG-INFORM message. This is an informational, optional mechanism, and is designed to aid in troubleshooting and forensics. On its own, it is not intended to be a strong security access mechanism. In particular, the ADDR-REG-INFORM message MUST NOT be used for authentication and authorization purposes, because in addition to the reasons above, the packets containing the message may be dropped.

7. IANA Considerations

This document introduces the following new entities which require an allocation out of the DHCPv6 registries defined at http://www.iana.org/assignments/dhcpv6-parameters/:

* one new DHCPv6 option, described in Section 4.1 which requires an allocation out of the registry of DHCPv6 Option Codes:
  - Value: TBA0
  - Description: OPTION_ADDR_REG_ENABLE
  - Client ORO: Yes
  - Singleton Option: Yes
* two new DHCPv6 messages which require an allocation out of the registry of Message Types:
  - ADDR-REG-INFORM message (TBA1) described in Section 4.2
  - ADDR-REG-REPLY (TBA2) described in Section 4.3.

8. References

8.1. Normative References

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

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, DOI 10.17487/RFC4007, March 2005, <https://www.rfc-editor.org/rfc/rfc4007>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/rfc/rfc4193>.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, DOI 10.17487/RFC4704, October 2006, <https://www.rfc-editor.org/rfc/rfc4704>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/rfc/rfc4862>.

[RFC6939] Halwasia, G., Bhandari, S., and W. Dec, "Client Link-Layer Address Option in DHCPv6", RFC 6939, DOI 10.17487/RFC6939, May 2013, <https://www.rfc-editor.org/rfc/rfc6939>.

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

[RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/rfc/rfc8415>.

[RFC9131] Linkova, J., "Gratuitous Neighbor Discovery: Creating Neighbor Cache Entries on First-Hop Routers", RFC 9131, DOI 10.17487/RFC9131, October 2021, <https://www.rfc-editor.org/rfc/rfc9131>.

8.2. Informative References

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/rfc/rfc4861>.

[RFC6620] Nordmark, E., Bagnulo, M., and E. Levy-Abegnoli, "FCFS SAVI: First-Come, First-Served Source Address Validation Improvement for Locally Assigned IPv6 Addresses", RFC 6620, DOI 10.17487/RFC6620, May 2012, <https://www.rfc-editor.org/rfc/rfc6620>.

Acknowledgments

Many thanks to Bernie Volz for significant review and feedback, as well as Hermin Anggawijaya, Brian Carpenter, Stuart Cheshire, Alan DeKok, Ryan Globus, Erik Kline, David Lamparter, Ted Lemon, Eric Levy-Abegnoli, Aditi Patange, Jim Reid, Michael Richardson, Mark Smith, Eric Vyncke, Timothy Winters for their feedback, comments and guidance. We apologize if we inadvertently forgot to acknowledge anyone's contributions.

This document borrows heavily from a previous document, draft-ietf-dhc-addr-registration, which defined "a mechanism to register self-generated and statically configured addresses in DNS through a DHCPv6 server". That document was written Sheng Jiang, Gang Chen, Suresh Krishnan, and Rajiv Asati.

Contributors

Gang Chen
China Mobile
53A, Xibianmennei Ave.
Xuanwu District
Beijing
P.R. China
Email: phdgang@gmail.com

Authors' Addresses

Warren Kumari
Google, LLC
Email: warren@kumari.net

Suresh Krishnan
Cisco Systems, Inc.
Email: suresh.krishnan@gmail.com

Rajiv Asati
Independent
Email: rajiv.asati@gmail.com

Lorenzo Colitti
Google, LLC
Shibuya 3-21-3,
Japan
Email: lorenzo@google.com

Jen Linkova
Google, LLC
1 Darling Island Rd
Pyrmont 2009
Australia
Email: furry13@gmail.com

Sheng Jiang
Beijing University of Posts and Telecommunications
No. 10 Xitucheng Road
Beijing
Haidian District, 100083
China
Email: shengjiang@bupt.edu.cn

DNSOP Working Group
Internet-Draft
Updates: 7344, 8078 (if approved)
Intended status: Standards Track
Expires: 13 October 2024

P. Thomassen
deSEC, Secure Systems Engineering
N. Wisiol
deSEC, Technische Universität Berlin

11 April 2024

Automatic DNSSEC Bootstrapping using Authenticated Signals from the Zone's Operator
draft-ietf-dnsop-dnssec-bootstrapping-08

Abstract

This document introduces an in-band method for DNS operators to publish arbitrary information about the zones they are authoritative for, in an authenticated fashion and on a per-zone basis. The mechanism allows managed DNS operators to securely announce DNSSEC key parameters for zones under their management, including for zones that are not currently securely delegated.

Whenever DS records are absent for a zone's delegation, this signal enables the parent's registry or registrar to cryptographically validate the CDS/CDNSKEY records found at the child's apex. The parent can then provision DS records for the delegation without resorting to out-of-band validation or weaker types of cross-checks such as "Accept after Delay".

This document deprecates the DS enrollment methods described in Section 3 of RFC 8078 in favor of Section 4 of this document, and also updates RFC 7344.

[Ed note: This document is being collaborated on at https://github.com/desec-io/draft-ietf-dnsop-dnssec-bootstrapping/ (https://github.com/desec-io/draft-ietf-dnsop-dnssec-bootstrapping/). The authors gratefully accept pull requests.]

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/.

Copyright Notice

Table of Contents

1. Introduction

Securing a DNS delegation for the first time requires that the child's DNSSEC parameters be conveyed to the parent through some trusted channel. While the communication conceptually has to occur between the parent registry and the DNSSEC key holder, what exactly that means and how the communication is coordinated traditionally depends on the relationship the child has with the parent.

A typical situation is that the key is held by the child DNS operator; the communication thus often involves this entity. In addition, depending on the circumstances, it may also involve the Registrar, possibly via the Registrant (for details, see [RFC7344], Appendix A).

As observed in [RFC7344], these dependencies often result in a manual process that is susceptible to mistakes and/or errors. In addition, due to the annoyance factor of the process, involved parties may avoid the process of getting a DS record set published in the first place.

To alleviate these problems, automated provisioning of DS records has been specified in ([RFC8078]). It is based on the parental agent (registry or registrar) fetching DNSSEC key parameters from the CDS and CDNSKEY records ([RFC7344]) located at the child zone's apex, and validating them somehow. This validation can be done using the child's existing DNSSEC chain of trust if the objective is to update an existing DS record set (such as during key rollover). However, when bootstrapping a DNSSEC delegation, the child zone has no existing DNSSEC validation path, and other means to ensure the CDS/CDNSKEY records' legitimacy must be found.

Due to the lack of a comprehensive DNS-innate solution, either out-of-band methods have been used so far to complete the chain of trust, or cryptographic validation has been entirely dispensed with, in exchange for weaker types of cross-checks such as "Accept after Delay" ([RFC8078] Section 3.3). [RFC8078] does not define an in-band validation method for enabling DNSSEC.

This document aims to close this gap by introducing an in-band method for DNS operators to publish arbitrary information about the zones they are authoritative for, in an authenticated manner and on a per-zone basis. The mechanism allows managed DNS operators to securely announce DNSSEC key parameters for zones under their management. The

parent can then use this signal to cryptographically validate the CDS/CDNSKEY records found at an insecure child zone's apex and, upon success, secure the delegation.

While applicable to the vast majority of domains, the protocol does not support certain edge cases, such as excessively long child zone names, or DNSSEC bootstrapping for domains with in-bailick nameservers only (see Section 4.4).

DNSSEC bootstrapping is just one application of the generic signaling mechanism specified in this document. Other applications might arise in the future, such as publishing operational metadata or auxiliary information which the DNS operator likes to make known (e.g., API endpoints for third-party interaction).

Readers are expected to be familiar with DNSSEC, including [RFC4033], [RFC4034], [RFC4035], [RFC6781], [RFC7344], and [RFC8078].

1.1. Terminology

This section defines the terminology used in this document.

CDS/CDNSKEY This notation refers to CDS and/or CDNSKEY, i.e., one or both.

Child see [RFC8499] Section 7

Child DNS operator The entity that maintains and publishes the zone information for the child DNS.

Parent see [RFC8499] Section 7

Parental agent The entity that has the authority to insert DS records into the parent zone on behalf of the child. (It could be the registry, registrar, a reseller, or some other authorized entity.)

Signaling domain A hostname from the child's NS record set, prefixed with the label _signal. There are as many signaling domains as there are distinct NS targets.

Signaling name The labels that are prefixed to a signaling domain in order to identify a signaling type and a child zone's name (see Section 3.2).

Signaling record A DNS record located at a signaling name under a signaling domain. Signaling records are used by the child DNS operator to publish information about the child.

Signaling type A signal type identifier, such as _dsboot for DNSSEC bootstrapping.

Signaling zone The zone which is authoritative for a given signaling record.

1.2. Requirements Notation

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.

2. Updates to RFCs

The DS enrollment methods described in Section 3 of [RFC8078] are deprecated and SHOULD NOT be used. Child DNS operators and parental agents who wish to use CDS/CDNSKEY records for initial DS enrollment SHOULD instead support the authentication protocol described in Section 4 of this document.

In order to facilitate publication of signaling records for the purpose of DNSSEC bootstrapping (see Section 4.1), the first bullet ("Location") of [RFC7344] Section 4.1 is removed.

3. Signaling

This section describes the general mechanism by which a child DNS operator can publish an authenticated signal about a child zone. Parental agents (or any other party) can then discover and process the signal. Authenticity is ensured through standard DNSSEC validation.

3.1. Chain of Trust

If a child DNS operator implements the protocol, each signaling zone MUST be signed and be validatable by the parental agent (i.e. have a valid DNSSEC chain of trust). This is typically achieved by securely delegating each signaling zone.

For example, when publishing a signal that relates to a child zone with NS records ns1.example.net and ns2.example.org, the child DNS operator needs to ensure that the parental agent has a valid DNSSEC chain of trust for the zone(s) that are authoritative for the signaling domains _signal.ns1.example.net and _signal.ns2.example.org.

3.2. Signaling Names

To publish information about the child zone in an authenticated fashion, the child DNS operator MUST publish one or more signaling records at a signaling name under each signaling domain.

Signaling records MUST be accompanied by RRSIG records created with the corresponding signaling zone's key(s). The type and contents of these signaling records depend on the type of signal.

The signaling name identifies the child and the signaling type. It is identical to the child name (with the final root label removed), prefixed with a label containing the signaling type.

4. Bootstrapping a DNSSEC Delegation

When the child zone's CDS/CDNSKEY RRsets are used for setting up initial trust, they need to be authenticated. This is achieved by co-publishing the child's CDS/CDNSKEY records as an authenticated signal as described in Section 3. The parent can discover and validate it, thus transferring trust from the child DNS operator nameservers' chain of trust to the child zone.

This protocol is not intended for updating an existing DS RRset. For this purpose, the parental agent can validate the child's CDS/CDNSKEY records directly, using the chain of trust established by the existing DS RRset ([RFC7344] Section 4).

4.1. Signaling Consent to Act as the Child's Signer

To confirm its willingness to act as the child's delegated signer and authenticate the child's CDS/CDNSKEY RRsets, the child DNS operator MUST co-publish them at the corresponding signaling name under each out-of-bailiwick signaling domain (Section 3.2). For simplicity, the child DNS operator MAY also co-publish the child's CDS/CDNSKEY RRsets under signaling domains that are in bailiwick, although those signaling domains are not used for validation (Section 4.2).

Unlike the CDS/CDNSKEY records at the child's apex, signaling records MUST be signed with the corresponding signaling zone's key(s). Their contents MUST be identical to the corresponding records published at the child's apex.

Existing use of CDS/CDNSKEY records was specified at the child apex only ([RFC7344], Section 4.1). This protocol extends the use of these record types to non-apex owner names for the purpose of DNSSEC bootstrapping. To exclude the possibility of semantic collision, there MUST NOT be a zone cut at a signaling name.

To avoid relying on the benevolence of a single signaling domain parent (such as the corresponding TLD registry), it is RECOMMENDED to diversify the path from the root to the child's nameserver hostnames. This is best achieved by using different and independently operated TLDs for each one.

4.1.1. Example

For the purposes of bootstrapping the child zone example.co.uk with NS records ns1.example.net, ns2.example.org, and ns3.example.co.uk, the required signaling domains are _signal.ns1.example.net and _signal.ns2.example.org.

In the zones containing these domains, the child DNS operator authenticates the CDS/CDNSKEY records found at the child's apex by co-publishing them at the names:

_dsboot.example.co.uk._signal.ns1.example.net
_dsboot.example.co.uk._signal.ns2.example.org

The records are accompanied by RRSIG records created using the key(s) of the respective signaling zone.

Publication of signaling records under the in-bailiwick domain _signal.ns3.example.co.uk is not required.

4.2. Validating CDS/CDNSKEY Records for DNSSEC Bootstrapping

To validate a child's CDS/CDNSKEY RRset for DNSSEC bootstrapping, the parental agent, knowing both the child zone name and its NS hostnames, MUST execute the following steps:

1. verify that the child is not currently securely delegated and that at least one of its nameservers is out of bailiwick;

2. query the CDS/CDNSKEY records at the child zone apex directly from each of the authoritative servers as determined by the delegation's NS record set (without caching);

3. query the CDS/CDNSKEY records located at the signaling name under each out-of-bailiwick signaling domain using a trusted DNS resolver and enforce DNSSEC validation;

4. check (separately by record type) that all record sets retrieved in Steps 2 and 3 have equal contents;

If the above steps succeed without error, the CDS/CDNSKEY records are successfully validated, and the parental agent can proceed with the publication of the DS record set under the precautions described in [RFC8078], Section 5.

However, the parental agent MUST abort the procedure if an error condition occurs, in particular:

* in Step 1: the child is already securely delegated or has in-bailiwick nameservers only;

* in Step 2: any failure during the retrieval of the CDS/CDNSKEY records located at the child apex from any of the authoritative nameservers;

* in Step 3: any failure to retrieve the CDS/CDNSKEY RRsets located at the signaling name under any signaling domain, including failure of DNSSEC validation, or unauthenticated data (AD bit not set);

* in Step 4: inconsistent responses (for at least one of the types), including a record set that is empty in one of Steps 2 or 3, but non-empty in the other.

4.2.1. Example

To verify the CDS/CDNSKEY records for the child example.co.uk, the parental agent (assuming that the child delegation's NS records are ns1.example.net, ns2.example.org, and ns3.example.co.uk)

1. checks that the child domain is not yet securely delegated;

2. queries CDS/CDNSKEY records for example.co.uk directly from ns1.example.net, ns2.example.org, and ns3.example.co.uk (without caching);

3. queries and validates the CDS/CDNSKEY records located at (see Section 3.2; ns3.example.co.uk is ignored because it is in bailiwick)

_dsboot.example.co.uk._signal.ns1.example.net
_dsboot.example.co.uk._signal.ns2.example.org

4. checks that the CDS/CDNSKEY record sets retrieved in Steps 2 and 3 agree across responses.

If all these steps succeed, the parental agent can proceed to publish a DS record set as indicated by the validated CDS/CDNSKEY records.

As in-bailiwick signaling names do not have a chain of trust at bootstrapping time, the parental agent does not consider them during validation. Consequently, if all NS hostnames are in bailiwick, validation cannot be completed, and DS records are not published.

4.3. Triggers

Parental agents SHOULD trigger the procedure described in Section 4.2 once one of the following conditions is fulfilled:

* The parental agent receives a new or updated NS record set for a child;

* The parental agent receives a notification indicating that the child wishes to have its CDS/CDNSKEY RRset processed;

* The parental agent encounters a signaling record during a proactive, opportunistic scan (e.g. daily queries of signaling records for some or all of its delegations);

* The parental agent encounters a signaling record during an NSEC walk or when parsing a signaling zone (e.g. when made available via AXFR by the child DNS operator);

* Any other condition as deemed appropriate by local policy.

Timer-based trigger mechanisms (such as scans) exhibit undesirable properties with respect to processing delay and scaling; on-demand triggers (like notifications) are preferable. Whenever possible, child DNS operators and parental agents are thus encouraged to use them, reducing both delays and the amount of scanning traffic.

Most types of discovery (such as daily scans of delegations) are based directly on the delegation's NS record set. In this case, these NS names can be used as is by the bootstrapping algorithm (Section 4.2) for querying signaling records.

Some discovery methods, however, do not imply reliable knowledge of the child's NS record set. For example, when discovering signaling names by performing an NSEC walk or zone transfer of a signaling zone, the parental agent MUST NOT assume that the nameserver(s) under whose signaling domain(s) a signaling name appears is actually authoritative for the corresponding child.

Instead, whenever a list of "bootstrappable domains" is obtained other than directly from the parent, the parental agent MUST ascertain that the child's delegation actually contains the nameserver hostname seen during discovery, and ensure that signaling record queries are only made against the proper set of nameservers as listed in the child's delegation from the parent.

4.4. Limitations

As a consequence of Step 3 in Section 4.2, DS bootstrapping does not work for fully in-bailiwick delegations, as no pre-existing chain of trust to the child domain is available during bootstrapping. (As a workaround, one can add an out-of-bailiwick nameserver to the initial NS record set and remove it once bootstrapping is completed. Automation for this is available via CSYNC records, see [RFC7477].)

Fully qualified signaling names must by valid DNS names. Label count and length requirements for DNS names imply that the protocol does not work for unusually long child domain names or NS hostnames.

5. Operational Recommendations

5.1. Child DNS Operator

CDS/CDNSKEY records and corresponding signaling records MUST NOT be published without the zone owner's consent. Likewise, the child DNS operator MUST enable the zone owner to signal the desire to turn off DNSSEC by publication of the special-value CDS/CDNSKEY RRset specified in [RFC8078] Section 4. To facilitate transitions between DNS operators, child DNS operators SHOULD support the multi-signer protocols described in [RFC8901].

Signaling domains SHOULD be delegated as standalone zones, so that the signaling zone's apex coincides with the signaling domain (such as _signal.ns1.example.net). While it is permissible for the signaling domain to be contained in a signaling zone of fewer labels (such as example.net), a zone cut ensures that bootstrapping activities do not require modifications of the zone containing the nameserver hostname.

Once a Child DNS Operator determines that specific signaling records have been processed (e.g., by seeing the result in the parent zone), they are advised to remove them. This will reduce the size of the Signaling Zone, and facilitate more efficient bulk processing (such as via zone transfers).

5.2. Parental Agent

In order to ensure timely DNSSEC bootstrapping of insecure domains, stalemate situations due to mismatch of cached records (Step 4 of Section 4.2) need to be avoided. It is thus RECOMMENDED to perform queries into signaling domains with an (initially) cold resolver cache, or to disable caching for them (e.g., by limiting response TTLs to the interval between scans).

6. Security Considerations

The DNSSEC bootstrapping method introduced in this document is based on the (now deprecated) approaches described in [RFC8078] Section 3, but adds authentication to the CDS/CDNSKEY concept. Its security level is therefore strictly higher than that of existing approaches described in that document (e.g. "Accept after Delay"). Apart from this general improvement, the same Security Considerations apply as in [RFC8078].

The level of rigor in Section 4.2 is needed to prevent publication of a half-baked DS RRset (authorized only under a subset of NS hostnames). This ensures, for example, that an operator in a multi-homed setup cannot enable DNSSEC unless all other operators agree.

In any case, as the child DNS operator has authoritative knowledge of the child's CDS/CDNSKEY records, it can readily detect fraudulent provisioning of DS records.

In order to prevent the TLD of nameserver hostnames from becoming a single point of failure for a delegation (both in terms of resolution availability and for the trust model of this protocol), it is advisable to use NS hostnames that are independent from each other with respect to their TLD.

7. IANA Considerations

Per [RFC8552], IANA is requested to add the following entries to the "Underscored and Globally Scoped DNS Node Names" registry:

RR Type	_NODE NAME	Reference
CDS	_signal	[draft-ietf-dnsop-dnssec-bootstrapping]
CDNSKEY	_signal	[draft-ietf-dnsop-dnssec-bootstrapping]

8. Implementation Status

Note to the RFC Editor: please remove this entire section before publication.

In addition to the information in this section, deployment is tracked by the community at https://github.com/oskar456/cds-updates (https://github.com/oskar456/cds-updates).

8.1. Child DNS Operator-side

* Operator support:
  - Cloudflare has implemented bootstrapping record synthesis for all signed customer zones.
  - Glauca HexDNS publishes bootstrapping records for its customer zones.
  - deSEC performs bootstrapping record synthesis for its zones using names _signal.ns1.desec.io and _signal.ns2.desec.org.
* Authoritative nameserver support:
  - Knot DNS supports signaling record synthesis since version 3.3.5.
  - An implementation of bootstrapping record synthesis in PowerDNS is available at https://github.com/desec-io/desec-ns/pull/46 (https://github.com/desec-io/desec-ns/pull/46).

8.2. Parental Agent-side

* ccTLD:
  - SWITCH (.ch, .li) has implemented authentication of consumed CDS records based on this draft.
  - .cl is working on an implementation.
* gTLD:
  - Knipp has implemented consumption of DNSSEC bootstrapping records in its TANGO and CORE registry systems.
  - A deployment of this is running at .swiss.
* Registrars:

  - Glauca has implemented authenticated CDS processing.

  - GoDaddy is working on an implementation.

* A tool to retrieve and process signaling records for bootstrapping purposes, either directly or via zone walking, is available at https://github.com/desec-io/dsbootstrap (https://github.com/desec-io/dsbootstrap). The tool outputs the validated DS records which then can be added to the parent zone.

9. Acknowledgements

Thanks to Brian Dickson, Ondej Caletka, John R. Levine, Christian Elmerot, Oli Schacher, Donald Eastlake, Libor Peltan, Warren Kumari, Scott Rose, Linda Dunbar, Tim Wicinski, Paul Wouters, Paul Hoffman for reviewing draft proposals and offering comments and suggestions.

Thanks also to Steve Crocker, Hugo Salgado, and Ulrich Wisser for early-stage brainstorming.

10. 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>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <https://www.rfc-editor.org/info/rfc4035>.

[RFC7344] Kumari, W., Gudmundsson, O., and G. Barwood, "Automating DNSSEC Delegation Trust Maintenance", RFC 7344, DOI 10.17487/RFC7344, September 2014, <https://www.rfc-editor.org/info/rfc7344>.

[RFC7477] Hardaker, W., "Child-to-Parent Synchronization in DNS", RFC 7477, DOI 10.17487/RFC7477, March 2015, <https://www.rfc-editor.org/info/rfc7477>.

[RFC8078] Gudmundsson, O. and P. Wouters, "Managing DS Records from the Parent via CDS/CDNSKEY", RFC 8078, DOI 10.17487/RFC8078, March 2017, <https://www.rfc-editor.org/info/rfc8078>.

[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>.

[RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/info/rfc8499>.

[RFC8552] Crocker, D., "Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves", BCP 222, RFC 8552, DOI 10.17487/RFC8552, March 2019, <https://www.rfc-editor.org/info/rfc8552>.

11. Informative References

[RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <https://www.rfc-editor.org/info/rfc6781>.

[RFC8901] Huque, S., Aras, P., Dickinson, J., Vcelak, J., and D. Blacka, "Multi-Signer DNSSEC Models", RFC 8901, DOI 10.17487/RFC8901, September 2020, <https://www.rfc-editor.org/info/rfc8901>.

Appendix A. Change History (to be removed before publication)

* draft-ietf-dnsop-dnssec-bootstrapping-08

  Editorial changes from AD Review

  Updated implementation section

  Change capitalization of terms from terminology section

* draft-ietf-dnsop-dnssec-bootstrapping-07

  Add Glauca registrar implementation

Editorial changes to Security Considerations

Add/discuss on-demand triggers (notifications)

* draft-ietf-dnsop-dnssec-bootstrapping-06

Add section "Updates to RFCs"

Editorial nits

Editorial changes from Secdir early review

* draft-ietf-dnsop-dnssec-bootstrapping-05

Editorial changes

* draft-ietf-dnsop-dnssec-bootstrapping-04

Added consent considerations.

Editorial changes.

* draft-ietf-dnsop-dnssec-bootstrapping-03

Updated Implementation section.

Typo fix.

* draft-ietf-dnsop-dnssec-bootstrapping-02

Clarified that RFC 8078 Section 3 is not replaced, but its methods are deprecated.

Added new deployments to Implementation section.

Included NSEC walk / AXFR as possible triggers for DS bootstrapping.

Editorial changes.

* draft-ietf-dnsop-dnssec-bootstrapping-01

Allow bootstrapping when some (not all) NS hostnames are in bailiwick.

Clarified Operational Recommendations according to operator feedback.

Turn loose Security Considerations points into coherent text.

Do no longer suggest NSEC-walking Signaling Domains. (It does not work well due to the Signaling Type prefix. What's more, it's unclear who would do this: Parents know there delegations and can do a targeted scan; others are not interested.)

Editorial changes.

Added IANA request.

Introduced Signaling Type prefix (_dsboot), renamed Signaling Name infix from _dsauth to _signal.

* draft-ietf-dnsop-dnssec-bootstrapping-00

Editorial changes.

* draft-thomassen-dnsop-dnssec-bootstrapping-03

Clarified importance of record cleanup by moving paragraph up.

Pointed out limitations.

Replace [RFC8078] Section 3 with our Section 4.2.

Changed _boot label to _dsauth.

Removed hashing of Child name components in Signaling Names.

Editorial changes.

* draft-thomassen-dnsop-dnssec-bootstrapping-02

Reframed as an authentication mechanism for RFC 8078.

Removed multi-signer use case (focus on RFC 8078 authentication).

Triggers need to fetch NS records (if not implicit from context).

Improved title.

Recognized that hash collisions are dealt with by Child apex check.

* draft-thomassen-dnsop-dnssec-bootstrapping-01

Add section on Triggers.

| Clarified title.
| Improved abstract.
| Require CDS/CDNSKEY records at the Child.
| Reworked Signaling Name scheme.
| Recommend using cold cache for consumption.
| Updated terminology (replace "Bootstrapping" by "Signaling").
| Added NSEC recommendation for Bootstrapping Zones.
| Added multi-signer use case.
| Editorial changes.

* draft-thomassen-dnsop-dnssec-bootstrapping-00

| Initial public draft.

Authors' Addresses

Peter Thomassen
deSEC, Secure Systems Engineering
Berlin
Germany
Email: peter@desec.io

Nils Wisiol
deSEC, Technische Universität Berlin
Berlin
Germany
Email: nils@desec.io

Network Working Group
Internet-Draft
Obsoletes: 5322 (if approved)
Updates: 3864, 4021 (if approved)
Intended status: Standards Track
Expires: 10 October 2024

P. Resnick, Ed.
Episteme

8 April 2024

Internet Message Format
draft-ietf-emailcore-rfc5322bis-11

Abstract

This document specifies the Internet Message Format (IMF), a syntax for text messages that are sent between computer users, within the framework of "electronic mail" messages. This specification is a revision of Request For Comments (RFC) 5322, itself a revision of Request For Comments (RFC) 2822, all of which supersede Request For Comments (RFC) 822, "Standard for the Format of ARPA Internet Text Messages", updating it to reflect current practice and incorporating incremental changes that were specified in other RFCs.

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 10 October 2024.

Copyright Notice

Table of Contents

1. Introduction

1.1. Scope

This document specifies the Internet Message Format (IMF), a syntax for text messages that are sent between computer users, within the framework of "electronic mail" messages. This specification is an update to [RFC5322], itself a revision of [RFC2822], all of which supersede [RFC0822], updating it to reflect current practice and incorporating incremental changes that were specified in other RFCs such as [RFC1123].

This document specifies a syntax only for text messages. In particular, it makes no provision for the transmission of images, audio, or other sorts of structured data in electronic mail messages. There are several extensions published, such as the MIME document series ([RFC2045], [RFC2046], [RFC2049]), which describe mechanisms for the transmission of such data through electronic mail, either by extending the syntax provided here or by structuring such messages to conform to this syntax. Those mechanisms are outside of the scope of this specification.

In the context of electronic mail, messages are viewed as having an envelope and contents. The envelope contains whatever information is needed to accomplish transmission and delivery. (See [I-D.ietf-emailcore-rfc5321bis] for a discussion of the envelope.) The contents comprise the object to be delivered to the recipient. This specification applies only to the format and some of the semantics of message contents. It contains no specification of the information in the envelope.

However, some message systems may use information from the contents to create the envelope. It is intended that this specification facilitate the acquisition of such information by programs.

This specification is intended as a definition of what message content format is to be passed between systems. Though some message systems locally store messages in this format (which eliminates the need for translation between formats) and others use formats that differ from the one specified in this specification, local storage is outside of the scope of this specification.

> Note: This specification is not intended to dictate the internal formats used by sites, the specific message system features that they are expected to support, or any of the characteristics of user interface programs that create or read messages. In addition, this document does not specify an encoding of the characters for either transport or storage;

> that is, it does not specify the number of bits used or how those bits are specifically transferred over the wire or stored on disk.

1.2. Notational Conventions

1.2.1. Requirements Notation

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 [BCP14] RFC2119 RFC8174 when, and only when, they appear in all capitals, as shown here.

1.2.2. Syntactic Notation

This specification uses the Augmented Backus-Naur Form (ABNF) [STD68] notation for the formal definitions of the syntax of messages. Characters will be specified either by a decimal value (e.g., the value %d65 for uppercase A and %d97 for lowercase A) or by a case-insensitive literal value enclosed in quotation marks (e.g., "A" for either uppercase or lowercase A).

1.2.3. Structure of This Document

This document is divided into several sections.

This section, section 1, is a short introduction to the document.

Section 2 lays out the general description of a message and its constituent parts. This is an overview to help the reader understand some of the general principles used in the later portions of this document. Any examples in this section MUST NOT be taken as specification of the formal syntax of any part of a message.

Section 3 specifies formal ABNF rules for the structure of each part of a message (the syntax) and describes the relationship between those parts and their meaning in the context of a message (the semantics). That is, it lays out the actual rules for the structure of each part of a message (the syntax) as well as a description of the parts and instructions for their interpretation (the semantics). This includes analysis of the syntax and semantics of subparts of messages that have specific structure. The syntax included in section 3 represents messages as they MUST be created. There are also notes in section 3 to indicate if any of the options specified in the syntax SHOULD be used over any of the others.

Both sections 2 and 3 describe messages that are legal to generate for purposes of this specification.

Section 4 of this document specifies an "obsolete" syntax. There are references in section 3 to these obsolete syntactic elements. The rules of the obsolete syntax are elements that have appeared in earlier versions of this specification or have previously been widely used in Internet messages. As such, these elements MUST be interpreted by parsers of messages in order to be conformant to this specification. However, since items in this syntax have been determined to be non-interoperable or to cause significant problems for recipients of messages, they MUST NOT be generated by creators of conformant messages.

> Note: The dictionary definition of "obsolete" is "no longer in use or no longer useful". While this specification mandates that these syntactic elements no longer be generated, it also mandates that conformant parsers be able to support them. One reason for this latter requirement is that there are long-established sites on the Internet with mail archives that go back decades, archives with messages containing these elements. Similarly, many people have decades-old messages in their personal message stores, and for various reasons it is occasionally useful to not only read such messages but also resend or forward them to others. While these archives may only be mined occasionally, and messages from these personal stores rarely resent, they are nonetheless still in use, making "obsolete" the incorrect term to describe these elements.
>
> Later efforts to revise this specification contemplated changing the term to "legacy" or something that would more accurately describe the elements, but such a change was rejected due to fears that it would result in unnecessary confusion, especially among long-time users and implementers of the specification.

Section 5 details security considerations to take into account when implementing this specification.

Appendix A lists examples of different sorts of messages. These examples are not exhaustive of the types of messages that appear on the Internet, but give a broad overview of certain syntactic forms.

Appendix B lists the differences between this specification and earlier specifications for Internet messages.

Appendix C contains acknowledgements.

2. Lexical Analysis of Messages

2.1. General Description

At the most basic level, a message is a series of characters. A message that is conformant with this specification is composed of characters with values in the range of 1 through 127 and interpreted as US-ASCII [ANSI.X3-4.1986] characters. For brevity, this document sometimes refers to this range of characters as simply "US-ASCII characters".

> Note: This document specifies that messages are made up of characters in the US-ASCII range of 1 through 127. There are other documents, specifically the MIME document series ([RFC2045], [RFC2046], [RFC2047], [RFC2049], [BCP13]) and the Internationalized Email Headers specification ([RFC6532]), that extend this specification to allow for values outside of that range. Discussion of those mechanisms is not within the scope of this specification.

Messages are divided into lines of characters. A line is a series of characters that is delimited with the two characters carriage-return and line-feed; that is, the carriage return (CR) character (ASCII value 13) followed immediately by the line feed (LF) character (ASCII value 10). (The carriage return/line feed pair is usually written in this document as "CRLF".)

A message consists of header fields (collectively called "the header section of the message") followed, optionally, by a body. The header section is a sequence of lines of characters with special syntax as defined in this specification. The body is simply a sequence of characters that follows the header section and is separated from the header section by an empty line (i.e., a line with nothing preceding the CRLF).

> Note: Common parlance and earlier versions of this specification use the term "header" to either refer to the entire header section or to refer to an individual header field. To avoid ambiguity, this document does not use the terms "header" or "headers" in isolation, but instead always uses "header field" to refer to the individual field and "header section" to refer to the entire collection.

2.1.1. Line Length Limits

There are two limits that this specification places on the number of characters in a line. Each line of characters MUST be no more than 998 characters, and SHOULD be no more than 78 characters, excluding the CRLF.

The 998 character limit is due to limitations in many implementations that send, receive, or store IMF messages which simply cannot handle more than 998 characters on a line. Receiving implementations would do well to handle an arbitrarily large number of characters in a line for robustness sake. However, there are so many implementations that (in compliance with the transport requirements of [I-D.ietf-emailcore-rfc5321bis]) do not accept messages containing more than 1000 characters including the CR and LF per line, it is important for implementations not to create such messages.

The more conservative 78 character recommendation is to accommodate the many implementations of user interfaces that display these messages which may truncate, or disastrously wrap, the display of more than 78 characters per line, in spite of the fact that such implementations are non-conformant to the intent of this specification (and that of [I-D.ietf-emailcore-rfc5321bis] if they actually cause information to be lost). Again, even though this limitation is put on messages, it is incumbent upon implementations that display messages to handle an arbitrarily large number of characters in a line (certainly at least up to the 998 character limit) for the sake of robustness.

2.2. Header Fields

Header fields are lines beginning with a field name, followed by a colon (":", ASCII value 58), followed by a field body, and terminated by CRLF. A field name MUST be composed of printable US-ASCII characters except for space (SP, ASCII value 32) (i.e., characters that have values between 33 and 126, inclusive) excluding colon. A field body may be composed of printable US-ASCII characters, including the space character, plus the horizontal tab (HTAB, ASCII value 9) character. (Together, SP and HTAB are known as the white space characters, WSP). A field body MUST NOT include CR and LF except when used in "folding" and "unfolding", as described in section 2.2.3. All field bodies MUST conform to the syntax described in sections 3 and 4 of this specification.

2.2.1. Unstructured Header Field Bodies

Some field bodies in this specification are defined simply as "unstructured" (which is specified in section 3.2.5 as any printable US-ASCII characters, including the space character, plus the horizontal tab character) with no further restrictions. These are referred to as unstructured field bodies. Semantically, unstructured field bodies are simply to be treated as a single line of characters with no further processing (except for "folding" and "unfolding" as described in section 2.2.3).

2.2.2. Structured Header Field Bodies

Some field bodies in this specification have a syntax that is more restrictive than the unstructured field bodies described above. These are referred to as "structured" field bodies. Structured field bodies are sequences of specific lexical tokens as described in sections 3 and 4 of this specification. Many of these tokens are allowed (according to their syntax) to be introduced or end with comments (as described in section 3.2.2) as well as the white space characters, and those white space characters are subject to "folding" and "unfolding" as described in section 2.2.3. Semantic analysis of structured field bodies is given along with their syntax.

2.2.3. Long Header Fields

Each header field is logically a single line of characters comprising the field name, the colon, and the field body. For convenience however, and to deal with the 998/78 character limitations per line, the field body portion of a header field can be split into a multiple-line representation; this is called "folding". The general rule is that wherever this specification allows for folding white space (not simply WSP characters), a CRLF may be inserted before any WSP.

For example, the header field:

```
Subject: This is a test
```

can be represented as:

```
Subject: This
 is a test
```

Though structured field bodies are defined in such a way that folding can take place between many of the lexical tokens (and even within some of the lexical tokens), folding SHOULD be limited to placing the CRLF at higher-level syntactic breaks. For instance, if a field body

is defined as comma-separated values, it is recommended that folding occur after the comma separating the structured items in preference to other places where the field could be folded, even if it is allowed elsewhere.

The process of moving from this folded multiple-line representation of a header field to its single line representation is called "unfolding". Unfolding is accomplished by simply removing any CRLF that is immediately followed by WSP. Each header field should be treated in its unfolded form for further syntactic and semantic evaluation. An unfolded header field has no length restriction and therefore may be indeterminately long.

2.3. Body

The body of a message is simply lines of US-ASCII characters. The only two limitations on the body are as follows:

* CR and LF MUST only occur together as CRLF; they MUST NOT appear independently in the body.

* Lines of characters in the body MUST be limited to 998 characters, and SHOULD be limited to 78 characters, excluding the CRLF.

> Note: As was stated earlier, there are other documents, specifically the MIME documents ([RFC2045], [RFC2046], [RFC2049], [BCP13]), that extend (and limit) this specification to allow for different sorts of message bodies. Again, these mechanisms are beyond the scope of this document.

3. Syntax

3.1. Introduction

The syntax as given in this section defines the legal syntax of Internet messages. Messages that are conformant to this specification MUST conform to the syntax in this section. If there are options in this section where one option SHOULD be generated, that is indicated either in the prose or in a comment next to the syntax.

For the defined expressions, a short description of the syntax and use is given, followed by the syntax in ABNF, followed by a semantic analysis. The following primitive tokens that are used but otherwise unspecified are taken from the "Core Rules" of [STD68], Appendix B.1: CR, LF, CRLF, HTAB, SP, WSP, DQUOTE, DIGIT, ALPHA, and VCHAR.

In some of the definitions, there will be non-terminals whose names start with "obs-". These "obs-" elements refer to tokens defined in the obsolete syntax in section 4. In all cases, these productions are to be ignored for the purposes of generating legal Internet messages and MUST NOT be used as part of such a message. However, when interpreting messages, these tokens MUST be honored as part of the legal syntax. In this sense, section 3 defines a grammar for the generation of messages, with "obs-" elements that are to be ignored, while section 4 adds grammar for the interpretation of messages.

3.2. Lexical Tokens

The following rules are used to define an underlying lexical analyzer, which feeds tokens to the higher-level parsers. This section defines the tokens used in structured header field bodies.

> Note: Readers of this specification need to pay special attention to how these lexical tokens are used in both the lower-level and higher-level syntax later in the document. Particularly, the white space tokens and the comment tokens defined in section 3.2.2 get used in the lower-level tokens defined here, and those lower-level tokens are in turn used as parts of the higher-level tokens defined later. Therefore, white space and comments may be allowed in the higher-level tokens even though they may not explicitly appear in a particular definition.

3.2.1. Quoted characters

Some characters are reserved for special interpretation, such as delimiting lexical tokens. To permit use of these characters as uninterpreted data, a quoting mechanism is provided.

```
quoted-pair     =   ("\" (VCHAR / WSP)) / obs-qp
```

Where any quoted-pair appears, it is to be interpreted as the character alone. That is to say, the "\" character that appears as part of a quoted-pair is semantically "invisible".

> Note: The "\" character may appear in a message where it is not part of a quoted-pair. A "\" character that does not appear in a quoted-pair is not semantically invisible. The only places in this specification where quoted-pair currently appears are ccontent, qcontent, and in obs-dtext in section 4.

3.2.2. Folding White Space and Comments

White space characters, including white space used in folding (described in section 2.2.3), may appear between many elements in header field bodies. Also, strings of characters that are treated as comments may be included in structured field bodies as characters enclosed in parentheses. The following defines the folding white space (FWS) and comment constructs.

Strings of characters enclosed in parentheses are considered comments so long as they do not appear within a "quoted-string", as defined in section 3.2.4. Comments may nest.

There are several places in this specification where comments and FWS may be freely inserted. To accommodate that syntax, an additional token for "CFWS" is defined for places where comments and/or FWS can occur. However, where CFWS occurs in this specification, it MUST NOT be inserted in such a way that any line of a folded header field is made up entirely of WSP characters and nothing else.

```
FWS             =   ([*WSP CRLF] 1*WSP) /  obs-FWS
                                        ; Folding white space

ctext           =   %d33-39 /           ; VCHAR characters not including
                    %d42-91 /           ;  "(", ")", or "\"
                    %d93-126 /
                    obs-ctext

ccontent        =   ctext / quoted-pair / comment

comment         =   "(" *([FWS] ccontent) [FWS] ")"

CFWS            =   (1*([FWS] comment) [FWS]) / FWS
```

Throughout this specification, where FWS (the folding white space token) appears, it indicates a place where folding, as discussed in section 2.2.3, may take place. Wherever folding appears in a message (that is, a header field body containing a CRLF followed by any WSP), unfolding (removal of the CRLF) is performed before any further semantic analysis is performed on that header field according to this specification. That is to say, any CRLF that appears in FWS is semantically "invisible".

A comment is normally used in a structured field body to provide some human-readable informational text. Since a comment is allowed to contain FWS, folding is permitted within the comment. Also note that since quoted-pair is allowed in a comment, the parentheses and backslash characters may appear in a comment, so long as they appear

as a quoted-pair. Semantically, the enclosing parentheses are not part of the comment; the comment is what is contained between the two parentheses. As stated earlier, the "\" in any quoted-pair and the CRLF in any FWS that appears within the comment are semantically "invisible" and therefore not part of the comment either.

Runs of FWS, comment, or CFWS that occur between lexical tokens in a structured header field are semantically interpreted as a single space character.

3.2.3. Atom

Several productions in structured header field bodies are simply strings of certain basic characters. Such productions are called atoms.

Some of the structured header field bodies also allow the period character (".", ASCII value 46) within runs of atext. An additional "dot-atom" token is defined for those purposes.

> Note: The "specials" token does not appear anywhere else in this specification. It is simply the VCHAR characters that do not appear in atext. It is provided only because it is useful for implementers who use tools that lexically analyze messages. Each of the characters in specials can be used to indicate a tokenization point in lexical analysis.

```
atext           =   ALPHA / DIGIT / ; VCHAR characters not including
                    "!" / "#" /     ;  specials. Used for atoms.
                    "$" / "%" /
                    "&" / "'" /
                    "*" / "+" /
                    "-" / "/" /
                    "=" / "?" /
                    "^" / "_" /
                    "`" / "{" /
                    "|" / "}" /
                    "~"

atom            =   [CFWS] 1*atext [CFWS]

dot-atom-text   =   1*atext *("." 1*atext)

dot-atom        =   [CFWS] dot-atom-text [CFWS]

specials        =   "(" / ")" /     ; Special characters that do
                    "<" / ">" /     ;  not appear in atext
                    "[" / "]" /
                    ":" / ";" /
                    "@" / "\" /
                    "," / "." /
                    DQUOTE
```

Both atom and dot-atom are interpreted as a single unit, comprising the string of characters that make it up. Semantically, the optional comments and FWS surrounding the rest of the characters are not part of the atom; the atom is only the run of atext characters in an atom, or the atext and "." characters in a dot-atom.

3.2.4. Quoted Strings

Strings of characters that include characters other than those allowed in atoms can be represented in a quoted string format, where the characters are surrounded by quote (DQUOTE, ASCII value 34) characters.

```
qtext           =   %d33 /          ; VCHAR characters not including
                    %d35-91 /       ;  "\" or the quote character
                    %d93-126 /
                    obs-qtext

qcontent        =   qtext / quoted-pair

quoted-string   =   [CFWS]
                    DQUOTE *([FWS] qcontent) [FWS] DQUOTE
                    [CFWS]
```

A quoted-string is treated as a unit. That is, quoted-string is identical to atom, semantically. Since a quoted-string is allowed to contain FWS, folding is permitted. Also note that since quoted-pair is allowed in a quoted-string, the quote and backslash characters may appear in a quoted-string so long as they appear as a quoted-pair.

Semantically, neither the optional CFWS outside of the quote characters nor the quote characters themselves are part of the quoted-string; the quoted-string is what is contained between the two quote characters. As stated earlier, the "\" in any quoted-pair and the CRLF in any FWS/CFWS that appears within the quoted-string are semantically "invisible" and therefore not part of the quoted-string either.

3.2.5. Miscellaneous Tokens

Three additional tokens are defined: word and phrase for combinations of atoms and/or quoted-strings, and unstructured for use in unstructured header fields and in some places within structured header fields.

```
word            =   atom / quoted-string

phrase          =   1*word / obs-phrase

unstructured    =   (*([FWS] VCHAR) *WSP) / obs-unstruct
```

3.3. Date and Time Specification

Date and time values occur in several header fields. This section specifies the syntax for a full date and time specification. Though folding white space is permitted throughout the date-time specification, it is RECOMMENDED that a single space be used in each place that FWS appears (whether it is required or optional); some older implementations will not interpret longer sequences of folding white space correctly.

```
   date-time       =   [ day-of-week "," ] date time [CFWS]

   day-of-week     =   ([FWS] day-name) / obs-day-of-week

   day-name        =   "Mon" / "Tue" / "Wed" / "Thu" /
                       "Fri" / "Sat" / "Sun"

   date            =   day month year

   day             =   ([FWS] 1*2DIGIT FWS) / obs-day

   month           =   "Jan" / "Feb" / "Mar" / "Apr" /
                       "May" / "Jun" / "Jul" / "Aug" /
                       "Sep" / "Oct" / "Nov" / "Dec"

   year            =   (FWS 4*DIGIT FWS) / obs-year

   time            =   time-of-day zone

   time-of-day     =   hour ":" minute [ ":" second ]

   hour            =   2DIGIT / obs-hour

   minute          =   2DIGIT / obs-minute

   second          =   2DIGIT / obs-second

   zone            =   (FWS ( "+" / "-" ) 4DIGIT) / obs-zone
```

The day is the numeric day of the month. The year is any numeric year 1900 or later.

The time-of-day specifies the number of hours, minutes, and optionally seconds since midnight of the date indicated (at the offset specified by the zone).

The date and time-of-day SHOULD express local time.

The zone specifies the offset from Coordinated Universal Time (UTC) that the date and time-of-day represent. The "+" or "-" indicates whether the time-of-day is ahead of (i.e., east of) or behind (i.e., west of) Universal Time. The first two digits indicate the number of hours difference from Universal Time, and the last two digits indicate the number of additional minutes difference from Universal Time. (Hence, +hhmm means +(hh * 60 + mm) minutes, and -hhmm means -(hh * 60 + mm) minutes). The form "+0000" SHOULD be used to indicate a time zone at Universal Time. Though "-0000" also indicates Universal Time, it is used to indicate that the time was

generated on a system that may be in a local time zone other than Universal Time and that the date-time contains no information about the local time zone.

A date-time specification MUST be semantically valid. That is, the day-of-week (if included) MUST be the day implied by the date, the numeric day-of-month MUST be between 1 and the number of days allowed for the specified month (in the specified year), the time-of-day MUST be in the range 00:00:00 through 23:59:60 (the number of seconds allowing for a leap second; see [RFC3339]), and the last two digits of the zone MUST be within the range 00 through 59.

3.4. Address Specification

Addresses occur in several message header fields to indicate senders and recipients of messages. An address may either be an individual mailbox, or a group of mailboxes.

```
address         =   mailbox / group

mailbox         =   name-addr / addr-spec

name-addr       =   [display-name] angle-addr

angle-addr      =   [CFWS] "<" addr-spec ">" [CFWS] /
                    obs-angle-addr

group           =   display-name ":" [group-list] ";" [CFWS]

display-name    =   phrase

mailbox-list    =   (mailbox *("," mailbox)) / obs-mbox-list

address-list    =   (address *("," address)) / obs-addr-list

group-list      =   mailbox-list / CFWS / obs-group-list
```

A mailbox receives mail. It is a conceptual entity that does not necessarily pertain to file storage. For example, some sites may choose to print mail on a printer and deliver the output to the addressee's desk.

Normally, a mailbox is composed of two parts: (1) an optional display name that indicates the name of the recipient (which can be a person or a system) that could be displayed to the user of a mail application, and (2) an addr-spec address enclosed in angle brackets ("<" and ">"). There is an alternate simple form of a mailbox where the addr-spec address appears alone, without the recipient's name or the angle brackets. The Internet addr-spec address is described in section 3.4.1.

Some legacy implementations used the simple form where the addr-spec appears without the angle brackets, but included the name of the recipient in parentheses as a comment following the addr-spec. Since the meaning of the information in a comment is unspecified, implementations SHOULD use the full name-addr form of the mailbox, instead of the legacy form, to specify the display name associated with a mailbox. Also, because some legacy implementations interpret the comment, comments generally SHOULD NOT be used in address fields to avoid confusing such implementations.

When it is desirable to treat several mailboxes as a single unit (i.e., in a distribution list), the group construct can be used. The group construct allows the sender to indicate a named group of recipients. This is done by giving a display name for the group, followed by a colon, followed by a comma-separated list of any number of mailboxes (including zero and one), and ending with a semicolon. Because the list of mailboxes can be empty, using the group construct is also a simple way to communicate to recipients that the message was sent to one or more named sets of recipients, without actually providing the individual mailbox address for any of those recipients.

3.4.1. Addr-Spec Specification

An addr-spec is a specific Internet identifier that contains a locally interpreted string followed by the at-sign character ("@", ASCII value 64) followed by an Internet domain. The locally interpreted string is either a quoted-string or a dot-atom. If the string can be represented as a dot-atom (that is, it contains no characters other than atext characters or one or more of "." surrounded by atext characters), then the dot-atom form SHOULD be used and the quoted-string form SHOULD NOT be used. Comments and folding white space SHOULD NOT be used around the "@" in the addr-spec.

> Note: A liberal syntax for the domain portion of addr-spec is given here. However, the domain portion contains addressing information specified by and used in other protocols (e.g., [STD13], [RFC1123], [I-D.ietf-emailcore-rfc5321bis]). It is therefore incumbent upon implementations to conform to the syntax of addresses for the context in which they are used.

```
addr-spec       =   local-part "@" domain

local-part      =   dot-atom / quoted-string / obs-local-part

domain          =   dot-atom / domain-literal / obs-domain

domain-literal  =   [CFWS] "[" *([FWS] dtext) [FWS] "]" [CFWS]

dtext           =   %d33-90 /          ; VCHAR characters not including
                    %d94-126 /         ;  "[", "]", or "\"
                    obs-dtext
```

The domain portion identifies the point to which the mail is delivered. In the dot-atom form, this is interpreted as an Internet domain name (either a host name or a mail exchanger name) as described in [STD13] and [RFC1123]. In the domain-literal form, the domain is interpreted as the literal Internet address of the particular host. In both cases, how addressing is used and how messages are transported to a particular host is covered in separate documents, such as [I-D.ietf-emailcore-rfc5321bis]. These mechanisms are outside of the scope of this document.

The local-part portion is a domain-dependent string. In addresses, it is simply interpreted on the particular host as a name of a particular mailbox.

3.5. Overall Message Syntax

A message consists of header fields, optionally followed by a message body. Lines in a message MUST be a maximum of 998 characters excluding the CRLF, but it is RECOMMENDED that lines be limited to 78 characters excluding the CRLF. (See section 2.1.1 for explanation.) In a message body, though all of the characters listed in the text rule MAY be used, the use of US-ASCII control characters (values 1 through 8, 11, 12, and 14 through 31) is discouraged since their interpretation by receivers for display is not guaranteed.

```
message         =   (fields / obs-fields)
                    [CRLF body]

body            =   (*(*998text CRLF) *998text) / obs-body

text            =   %d1-9 /          ; Characters excluding CR
                    %d11 /           ;  and LF
                    %d12 /
                    %d14-127
```

The header fields carry most of the semantic information and are defined in section 3.6. The body is simply a series of lines of text that are uninterpreted for the purposes of this specification.

3.6. Field Definitions

The header fields of a message are defined here. All header fields have the same general syntactic structure: a field name, followed by a colon, followed by the field body. The specific syntax for each header field is defined in the subsequent sections.

> Note: In the ABNF syntax for each field in subsequent sections, each field name is followed by the required colon. However, for brevity, sometimes the colon is not referred to in the textual description of the syntax. It is, nonetheless, required.

It is important to note that the header fields are not guaranteed to be in a particular order. They may appear in any order, and they have been known to be reordered occasionally when transported over the Internet. However, for the purposes of this specification, header fields SHOULD NOT be reordered when a message is transported or transformed. More importantly, the trace header fields and resent header fields MUST NOT be reordered, and SHOULD be kept in blocks prepended to the message. See sections 3.6.6 and 3.6.7 for more information.

The only required header fields are the origination date field and the originator address field(s). All other header fields are syntactically optional. More information is contained in the table following this definition.

```
fields          =   *(trace
                      *(resent-date /
                       resent-from /
                       resent-sender /
                       resent-to /
                       resent-cc /
                       resent-bcc /
                       resent-msg-id))
                    *(orig-date /
                    from /
                    sender /
                    reply-to /
                    to /
                    cc /
                    bcc /
                    message-id /
                    in-reply-to /
                    references /
                    subject /
                    comments /
                    keywords /
                    optional-field)
```

The following table indicates limits on the number of times each field may occur in the header section of a message as well as any special limitations on the use of those fields. An asterisk ("*") next to a value in the minimum or maximum column indicates that a special restriction appears in the Notes column.

Field	Min number	Max number	Notes
trace	0	unlimited	Block prepended - see 3.6.7
resent-date	0*	unlimited*	One per block, required if other resent fields are present - see 3.6.6
resent-from	0	unlimited*	One per block - see 3.6.6
resent-sender	0*	unlimited*	One per block, MUST occur with multi-address resent-from - see 3.6.6
resent-to	0	unlimited*	One per block - see

			3.6.6
resent-cc	0	unlimited*	One per block - see 3.6.6
resent-bcc	0	unlimited*	One per block - see 3.6.6
resent-msg-id	0	unlimited*	One per block - see 3.6.6
orig-date	1	1	
from	1	1	See sender and 3.6.2
sender	0*	1	MUST occur with multi-address from - see 3.6.2
reply-to	0	1	
to	0	1	
cc	0	1	
bcc	0	1	
message-id	0*	1	SHOULD be present - see 3.6.4
in-reply-to	0*	1	SHOULD occur in some replies - see 3.6.4
references	0*	1	SHOULD occur in some replies - see 3.6.4
subject	0	1	
comments	0	unlimited	
keywords	0	unlimited	
optional-field	0	unlimited	

Table 1

The exact interpretation of each field is described in subsequent sections.

3.6.1. The Origination Date Field

The origination date field consists of the field name "Date" followed by a date-time specification.

```
orig-date       =   "Date:" date-time CRLF
```

The origination date specifies the date and time at which the creator of the message indicated that the message was complete and ready to enter the mail delivery system. For instance, this might be the time that a user pushes the "send" or "submit" button in an application program. In any case, it is specifically not intended to convey the time that the message is actually transported, but rather the time at which the human or other creator of the message has put the message into its final form, ready for transport. (For example, a portable computer user who is not connected to a network might queue a message for delivery. The origination date is intended to contain the date and time that the user queued the message, not the time when the user connected to the network to send the message.)

3.6.2. Originator Fields

The originator fields of a message consist of the from field, the sender field (when applicable), and optionally the reply-to field. The from field consists of the field name "From" and a comma-separated list of one or more mailbox specifications. If the from field contains more than one mailbox specification in the mailbox-list, then the sender field, containing the field name "Sender" and a single mailbox specification, MUST appear in the message. In either case, an optional reply-to field MAY also be included, which contains the field name "Reply-To" and a comma-separated list of one or more addresses.

```
from            =   "From:" mailbox-list CRLF

sender          =   "Sender:" mailbox CRLF

reply-to        =   "Reply-To:" address-list CRLF
```

The originator fields indicate the mailbox(es) of the source of the message. The "From:" field specifies the author(s) of the message, that is, the mailbox(es) of the person(s) or system(s) responsible for the writing of the message. The "Sender:" field specifies the mailbox of the agent responsible for the actual transmission of the message. For example, if a secretary were to send a message for another person, the mailbox of the secretary would appear in the "Sender:" field and the mailbox of the actual author would appear in the "From:" field. If the originator of the message can be indicated

by a single mailbox and the author and transmitter are identical, the "Sender:" field SHOULD NOT be used. Otherwise, both fields SHOULD appear.

> Note: The transmitter information is always present. The absence of the "Sender:" field is sometimes mistakenly taken to mean that the agent responsible for transmission of the message has not been specified. This absence merely means that the transmitter is identical to the author and is therefore not redundantly placed into the "Sender:" field.

The originator fields also provide the information required when replying to a message. When the "Reply-To:" field is present, it indicates the address(es) to which the author of the message suggests that replies be sent. In the absence of the "Reply-To:" field, replies SHOULD by default be sent to the mailbox(es) specified in the "From:" field unless otherwise specified by the person composing the reply.

In all cases, the "From:" field SHOULD NOT contain any mailbox that does not belong to the author(s) of the message. See also section 3.6.3 for more information on forming the destination addresses for a reply.

3.6.3. Destination Address Fields

The destination fields of a message consist of three possible fields, each of the same form: the field name, which is either "To", "Cc", or "Bcc", followed by a comma-separated list of one or more addresses (either mailbox or group syntax).

```
to              =   "To:" address-list CRLF

cc              =   "Cc:" address-list CRLF

bcc             =   "Bcc:" [address-list / CFWS] CRLF
```

The destination fields specify the recipients of the message. Each destination field may have one or more addresses, and the addresses indicate the intended recipients of the message. The only difference between the three fields is how each is used.

The "To:" field contains the address(es) of the primary recipient(s) of the message.

The "Cc:" field (where the "Cc" means "Carbon Copy" in the sense of making a copy on a typewriter using carbon paper) contains the addresses of others who are to receive the message, though the content of the message may not be directed at them.

The "Bcc:" field (where the "Bcc" means "Blind Carbon Copy") contains addresses of recipients of the message whose addresses are not to be revealed to other recipients of the message. This is accomplished in one of three ways:

1. The "Bcc:" field and its contents are not included in the message that is sent, even though the message is sent to all of the recipients in all of the destination ("To:", "Cc:", and "Bcc:") fields.

2. The recipients in the "To:" and "Cc:" fields are sent a message that does not include the "Bcc:" field or its contents, and the recipients in the "Bcc:" field are sent a copy with an empty "Bcc:" field.

3. The recipients in the "To:" and "Cc:" fields are sent a message that does not include the "Bcc:" field or its contents, and the recipients in the "Bcc:" field are each sent a personalized copy of the message that includes a "Bcc:" field containing only their own address.

Which method to use with "Bcc:" fields is implementation dependent, but refer to the "Security Considerations" section (Section 5) of this document for a discussion of each.

When a message is a reply to another message, the mailboxes of the authors of the original message (the mailboxes in the "From:" field) or mailboxes specified in the "Reply-To:" field (if it exists) MAY appear in the "To:" field of the reply since these would normally be the primary recipients of the reply. If a reply is sent to a message that has destination fields, it is often desirable to send a copy of the reply to all of the recipients of the message, in addition to the author. When such a reply is formed, addresses in the "To:" and "Cc:" fields of the original message MAY appear in the "Cc:" field of the reply, since these are normally secondary recipients of the reply. If a "Bcc:" field is present in the original message, addresses in that field MAY appear in the "Bcc:" field of the reply, but they SHOULD NOT appear in the "To:" or "Cc:" fields.

> Note: Some mail applications have automatic reply commands that include the destination addresses of the original message in the destination addresses of the reply. How those reply commands behave is implementation dependent and is beyond the

> scope of this document. In particular, whether or not to include the original destination addresses when the original message had a "Reply-To:" field is not addressed here.

3.6.4. Identification Fields

Though listed as optional in the table (Table 1) in section 3.6, every message SHOULD have a "Message-ID:" field. Furthermore, reply messages SHOULD have "In-Reply-To:" and "References:" fields as appropriate and as described below.

The "Message-ID:" field contains a single unique message identifier. The "References:" and "In-Reply-To:" fields each contain one or more unique message identifiers, optionally separated by CFWS.

The message identifier (msg-id) syntax is a limited version of the addr-spec construct enclosed in the angle bracket characters, "<" and ">". Unlike addr-spec, this syntax only permits the dot-atom-text form on the left-hand side of the "@" and does not have internal CFWS anywhere in the message identifier.

> Note: As with addr-spec, a liberal syntax is given for the right-hand side of the "@" in a msg-id. However, later in this section, the use of a domain for the right-hand side of the "@" is RECOMMENDED. Again, the syntax of domain constructs is specified by and used in other protocols (e.g., [STD13], [RFC1123], [I-D.ietf-emailcore-rfc5321bis]). It is therefore incumbent upon implementations to conform to the syntax of addresses for the context in which they are used.

```
message-id      =   "Message-ID:" msg-id CRLF

in-reply-to     =   "In-Reply-To:" 1*msg-id CRLF

references      =   "References:" 1*msg-id CRLF

msg-id          =   [CFWS] "<" msg-id-internal ">" [CFWS]

msg-id-internal =   id-left "@" id-right

id-left         =   dot-atom-text / obs-id-left

id-right        =   dot-atom-text / no-fold-literal / obs-id-right

no-fold-literal =   "[" *dtext "]"
```

The "Message-ID:" field provides a unique message identifier that refers to a particular version of a particular message. The uniqueness of the message identifier is guaranteed by the host that generates it (see below). This message identifier is intended to be machine readable and not necessarily meaningful to humans. A message identifier pertains to exactly one version of a particular message; subsequent revisions to the message each receive new message identifiers.

> Note: There are many instances when messages are "changed", but those changes do not constitute a new instantiation of that message, and therefore the message would not get a new message identifier. For example, when messages are introduced into the transport system, they are often prepended with additional header fields such as trace fields (described in section 3.6.7) and resent fields (described in section 3.6.6). The addition of such header fields does not change the identity of the message and therefore the original "Message-ID:" field is retained. In all cases, it is the meaning that the sender of the message wishes to convey (i.e., whether this is the same message or a different message) that determines whether or not the "Message-ID:" field changes, not any particular syntactic difference that appears (or does not appear) in the message.

The "In-Reply-To:" and "References:" fields are used when creating a reply to a message. They hold the message identifier of the original message and the message identifiers of other messages (for example, in the case of a reply to a message that was itself a reply). The "In-Reply-To:" field may be used to identify the message (or messages) to which the new message is a reply, while the "References:" field may be used to identify a "thread" of conversation.

When creating a reply to a message, the "In-Reply-To:" and "References:" fields of the resultant message are constructed as follows:

The "In-Reply-To:" field will contain the contents of the "Message-ID:" field of the message to which this one is a reply (the "parent message"). If there is more than one parent message, then the "In-Reply-To:" field will contain the contents of all of the parents' "Message-ID:" fields. If there is no "Message-ID:" field in any of the parent messages, then the new message will have no "In-Reply-To:" field.

The "References:" field will contain the contents of the parent's "References:" field (if any) followed by the contents of the parent's "Message-ID:" field (if any). If the parent message does not contain

a "References:" field but does have an "In-Reply-To:" field containing a single message identifier, then the "References:" field will contain the contents of the parent's "In-Reply-To:" field followed by the contents of the parent's "Message-ID:" field (if any). If the parent has none of the "References:", "In-Reply-To:", or "Message-ID:" fields, then the new message will have no "References:" field.

> Note: Some implementations parse the "References:" field to display the "thread of the discussion". These implementations assume that each new message is a reply to a single parent and hence that they can walk backwards through the "References:" field to find the parent of each message listed there. Therefore, trying to form a "References:" field for a reply that has multiple parents is discouraged; how to do so is not defined in this document.

The message identifier (msg-id) itself MUST be a globally unique identifier for a message. The generator of the message identifier MUST guarantee that the msg-id is unique. There are several algorithms that can be used to accomplish this. Since the msg-id has a similar syntax to addr-spec (identical except that quoted strings, comments, and folding white space are not allowed), a good method is to put the domain name (or a domain literal IP address) of the host on which the message identifier was created on the right-hand side of the "@" (since domain names and IP addresses are normally unique), and put a combination of the current absolute date and time along with some other currently unique (perhaps sequential) identifier available on the system (for example, a process id number) on the left-hand side. Though other algorithms will work, it is RECOMMENDED that the right-hand side contain some domain identifier (either of the host itself or otherwise) such that the generator of the message identifier can guarantee the uniqueness of the left-hand side within the scope of that domain.

Semantically, the angle bracket characters are not part of the msg-id; the msg-id is what is contained between the two angle bracket characters.

3.6.5. Informational Fields

The informational fields are all optional. The "Subject:" and "Comments:" fields are unstructured fields as defined in section 2.2.1, and therefore may contain text or folding white space. The "Keywords:" field contains a comma-separated list of one or more words or quoted-strings.

```
subject         =   "Subject:" unstructured CRLF

comments        =   "Comments:" unstructured CRLF

keywords        =   "Keywords:" phrase *("," phrase) CRLF
```

These three fields are intended to have only human-readable content with information about the message. The "Subject:" field is the most common and contains a short string identifying the topic of the message. When used in a reply, the field body MAY start with the string "Re: " (an abbreviation of the Latin "in re", meaning "in the matter of") followed by the contents of the "Subject:" field body of the original message. If this is done, only one instance of the literal string "Re: " ought to be used since use of other strings or more than one instance can lead to undesirable consequences. The "Comments:" field contains any additional comments on the text of the body of the message. The "Keywords:" field contains a comma-separated list of important words and phrases that might be useful for the recipient.

3.6.6. Resent Fields

Resent fields SHOULD be added to any message that is reintroduced by a user into the transport system. A separate set of resent fields SHOULD be added each time this is done. All of the resent fields corresponding to a particular resending of the message SHOULD be grouped together. Each new set of resent fields is prepended to the message; that is, the most recent set of resent fields appears earlier in the message. No other fields in the message are changed when resent fields are added.

Each of the resent fields corresponds to a particular field elsewhere in the syntax. For instance, the "Resent-Date:" field corresponds to the "Date:" field and the "Resent-To:" field corresponds to the "To:" field. In each case, the syntax for the field body is identical to the syntax given previously for the corresponding field.

When resent fields are used, the "Resent-From:" and "Resent-Date:" fields MUST be present. The "Resent-Message-ID:" field SHOULD be present. "Resent-Sender:" SHOULD NOT be used if "Resent-Sender:" would be identical to "Resent-From:".

```
resent-date     =   "Resent-Date:" date-time CRLF

resent-from     =   "Resent-From:" mailbox-list CRLF

resent-sender   =   "Resent-Sender:" mailbox CRLF

resent-to       =   "Resent-To:" address-list CRLF

resent-cc       =   "Resent-Cc:" address-list CRLF

resent-bcc      =   "Resent-Bcc:" [address-list / CFWS] CRLF

resent-msg-id   =   "Resent-Message-ID:" msg-id CRLF
```

Resent fields are used to identify a message as having been reintroduced into the transport system by a user. The purpose of using resent fields is to have the message appear to the final recipient as if it were sent directly by the original sender, with all of the original fields remaining the same. Each set of resent fields correspond to a particular resending event. That is, if a message is resent multiple times, each set of resent fields gives identifying information for each individual time. Resent fields are strictly informational. They MUST NOT be used in the normal processing of replies or other such automatic actions on messages.

> Note: Reintroducing a message into the transport system and using resent fields is a different operation from "forwarding". "Forwarding" has two meanings: One sense of forwarding is that a mail reading program can be told by a user to forward a copy of a message to another person, making the forwarded message the body of the new message. A forwarded message in this sense does not appear to have come from the original sender, but is an entirely new message from the forwarder of the message. Forwarding may also mean that a mail transport program gets a message and forwards it on to a different destination for final delivery. Resent header fields are not intended for use with either type of forwarding.

The resent originator fields indicate the mailbox of the person(s) or system(s) that resent the message. As with the regular originator fields, there are two forms: a simple "Resent-From:" form, which contains the mailbox of the individual doing the resending, and the more complex form, when one individual (identified in the "Resent-Sender:" field) resends a message on behalf of one or more others (identified in the "Resent-From:" field).

When replying to a resent message, replies behave just as they would with any other message, using the original "From:", "Reply-To:", "Message-ID:", and other fields. The resent fields are only informational and MUST NOT be used in the normal processing of replies.

The "Resent-Date:" indicates the date and time at which the resent message is dispatched by the resender of the message. Like the "Date:" field, it is not the date and time that the message was actually transported.

The "Resent-To:", "Resent-Cc:", and "Resent-Bcc:" fields function identically to the "To:", "Cc:", and "Bcc:" fields, respectively, except that they indicate the recipients of the resent message, not the recipients of the original message.

The "Resent-Message-ID:" field provides a unique identifier for the resent message.

3.6.7. Trace Fields

The trace fields form a group of header fields that normally includes a "Return-Path:" field and/or one or more "Received:" fields or other trace-optional fields that are defined by other specifications as belonging within the trace fields grouping. The "Return-Path:" header field contains a pair of angle brackets that enclose an optional addr-spec. The "Received:" field contains a (possibly empty) list of tokens followed by a semicolon and a date-time specification. Each token must be a word, angle-addr, addr-spec, or a domain. The trace-optional fields follow the syntax of section 3.6.8. Other specifications may provide for particular trace fields and more precisely define their syntax and semantics. Those specifications include [I-D.ietf-emailcore-rfc5321bis] for the forms of "Received:" and "Return-path:" supplied by SMTP.

```
trace           =   [return]
                    *(received / trace-optional)

return          =   "Return-Path:" path CRLF

path            =   angle-addr / ([CFWS] "<" [CFWS] ">" [CFWS])

received        =   "Received:"
                    [1*received-token / CFWS] ";" date-time CRLF

received-token  =   word / angle-addr / addr-spec / domain

trace-optional  =   optional-field
```

The trace fields provide an audit trail of message handling history. They document actions taken as a message moves through the transport system. A full discussion of the Internet mail use of the "Return-Path:" and "Received:" trace fields is contained in [I-D.ietf-emailcore-rfc5321bis]; other specifications such as [I-D.ietf-emailcore-as] describe the use of other fields that are to be interpreted as trace fields. In particular, the operational behavior associated with any of the trace fields (how, when, and whether they are added to or even removed from messages as they are moved through the transport system) are not described here. For the purposes of this specification, the trace fields are strictly informational, and any formal interpretation of them is outside of the scope of this document.

3.6.8. Optional Fields

Fields may appear in messages that are otherwise unspecified in this document. They MUST conform to the syntax of an optional-field. This is a field name, made up of the VCHAR characters except colon, followed by a colon, followed by any text that conforms to the unstructured syntax.

The field names of any optional field MUST NOT be identical to any field name specified elsewhere in this document.

```
optional-field  =   field-name ":" unstructured CRLF

field-name      =   1*ftext         ; Limit to 77 characters to
                                    ;  stay within 78 char-per-
                                    ;  line recommendation


ftext           =   %d33-57 /       ; VCHAR characters not including
                    %d59-126        ;  ":".
```

For the purposes of this specification, any optional field is uninterpreted.

4. Obsolete Syntax

Earlier versions of this specification allowed for different (usually more liberal) syntax than is allowed in this version. Also, there have been syntactic elements used in messages on the Internet whose interpretations have never been documented. Though these syntactic forms MUST NOT be generated according to the grammar in section 3, they MUST be accepted and parsed by a conformant receiver. This section documents many of these syntactic elements. (See the note in Section 1.2.3 for an explanation of the term "obsolete".) Taking the

grammar in section 3 and adding the definitions presented in this section will result in the grammar to use for the interpretation of messages.

> Note: There are Internet messages that do not conform to even the additional syntax given in this section. The fact that a particular form does not appear in any section of this document is not justification for computer programs to crash or for malformed data to be irretrievably lost by any implementation. It is up to the implementation to deal with messages robustly.

One important difference between the obsolete (interpreting) and the current (generating) syntax is that in structured header field bodies (i.e., between the colon and the CRLF of any structured header field), white space characters, including folding white space, and comments could be freely inserted between any syntactic tokens. This allowed many complex forms that have proven difficult for some implementations to parse.

Another key difference between the obsolete and the current syntax is that the rule in section 3.2.2 regarding lines composed entirely of white space in comments and folding white space does not apply. See the discussion of folding white space in section 4.2 below.

Finally, certain characters that were formerly allowed in messages appear in this section. The NUL character (ASCII value 0) was once allowed, but is no longer for compatibility reasons. Similarly, US-ASCII control characters other than CR, LF, SP, and HTAB (ASCII values 1 through 8, 11, 12, 14 through 31, and 127) were allowed to appear in header field bodies. CR and LF were allowed to appear in messages other than as CRLF; this use is also shown here.

Other differences in syntax and semantics are noted in the following sections.

4.1. Miscellaneous Obsolete Tokens

These syntactic elements are used elsewhere in the obsolete syntax or in the main syntax. Bare CR, bare LF, and NUL are added to obs-qp, obs-body, and obs-unstruct. US-ASCII control characters are added to obs-qp, obs-unstruct, obs-ctext, and obs-qtext. The period character is added to obs-phrase. The obs-phrase-list provides for a (potentially empty) comma-separated list of phrases that may include "null" elements. That is, there could be two or more commas in such a list with nothing in between them, or commas at the beginning or end of the list.

> Note: The "period" (or "full stop") character (".") in obs-phrase is not a form that was allowed in earlier versions of this or any other specification. Period (nor any other character from specials) was not allowed in phrase because it introduced a parsing difficulty distinguishing between phrases and portions of an addr-spec (see section 4.4). It appears here because the period character is currently used in many messages in the display-name portion of addresses, especially for initials in names, and therefore must be interpreted properly.

```
obs-NO-WS-CTL   =   %d1-8 /          ; US-ASCII control
                    %d11 /           ;  characters that do not
                    %d12 /           ;  include the carriage
                    %d14-31 /        ;  return, line feed, and
                    %d127            ;  white space characters

obs-ctext       =   obs-NO-WS-CTL

obs-qtext       =   obs-NO-WS-CTL

obs-utext       =   %d0 / obs-NO-WS-CTL / VCHAR

obs-qp          =   "\" (%d0 / obs-NO-WS-CTL / LF / CR)

obs-body        =   *(%d0 / LF / CR / text)

obs-unstruct    =   *((*CR 1*(obs-utext / FWS)) / 1*LF) *CR

obs-phrase      =   word *(word / "." / CFWS)

obs-phrase-list =   [phrase / CFWS] *("," [phrase / CFWS])
```

Bare CR and bare LF appear in messages with two different meanings. In many cases, bare CR or bare LF are used improperly instead of CRLF to indicate line separators. In other cases, bare CR and bare LF are used simply as US-ASCII control characters with their traditional ASCII meanings.

4.2. Obsolete Folding White Space

In the obsolete syntax, any amount of folding white space MAY be inserted where the obs-FWS rule is allowed. This creates the possibility of having two consecutive "folds" in a line, and therefore the possibility that a line which makes up a folded header field could be composed entirely of white space.

```
obs-FWS         =   1*([CRLF] WSP)
```

4.3. Obsolete Date and Time

The syntax for the obsolete date format allows a 2 digit year in the date field and allows for a list of alphabetic time zone specifiers that were used in earlier versions of this specification. It also permits comments and folding white space between many of the tokens.

```
obs-day-of-week =   [CFWS] day-name [CFWS]

obs-day         =   [CFWS] 1*2DIGIT [CFWS]

obs-year        =   [CFWS] 2*DIGIT [CFWS]

obs-hour        =   [CFWS] 2DIGIT [CFWS]

obs-minute      =   [CFWS] 2DIGIT [CFWS]

obs-second      =   [CFWS] 2DIGIT [CFWS]

obs-zone        =   [CFWS] (
                    "UT" / "GMT" /  ; Universal Time
                                    ; North American UT
                                    ; offsets
                    "EST" / "EDT" / ; Eastern:  - 5/ - 4
                    "CST" / "CDT" / ; Central:  - 6/ - 5
                    "MST" / "MDT" / ; Mountain: - 7/ - 6
                    "PST" / "PDT" / ; Pacific:  - 8/ - 7
                                    ;
                    %d65-73 /       ; Military zones - "A"
                    %d75-90 /       ; through "I" and "K"
                    %d97-105 /      ; through "Z", both
                    %d107-122       ; upper and lower case
                    ) [CFWS]
```

Where a two or three digit year occurs in a date, the year is to be interpreted as follows: If a two digit year is encountered whose value is between 00 and 49, the year is interpreted by adding 2000, ending up with a value between 2000 and 2049. If a two digit year is encountered with a value between 50 and 99, or any three digit year is encountered, the year is interpreted by adding 1900.

In the obsolete time zone, "UT" and "GMT" are indications of "Universal Time" and "Greenwich Mean Time", respectively, and are both semantically identical to "+0000".

The remaining three character zones are the US time zones. The first letter, "E", "C", "M", or "P" stands for "Eastern", "Central", "Mountain", and "Pacific". The second letter is either "S" for "Standard" time, or "D" for "Daylight" (daylight saving or summer) time. Their interpretations are as follows:

```
EDT is semantically equivalent to -0400
EST is semantically equivalent to -0500
CDT is semantically equivalent to -0500
CST is semantically equivalent to -0600
MDT is semantically equivalent to -0600
MST is semantically equivalent to -0700
PDT is semantically equivalent to -0700
PST is semantically equivalent to -0800
```

The 1 character military time zones were defined in a non-standard way in [RFC0822] and are therefore unpredictable in their meaning. The original definitions of the military zones "A" through "I" are equivalent to "+0100" through "+0900", respectively; "K", "L", and "M" are equivalent to "+1000", "+1100", and "+1200", respectively; "N" through "Y" are equivalent to "-0100" through "-1200". respectively; and "Z" is equivalent to "+0000". However, because of the error in [RFC0822], they SHOULD all be considered equivalent to "-0000" unless there is out-of-band information confirming their meaning.

Other multi-character (usually between 3 and 5) alphabetic time zones have been used in Internet messages. Any such time zone whose meaning is not known SHOULD be considered equivalent to "-0000" unless there is out-of-band information confirming their meaning.

4.4. Obsolete Addressing

There are four primary differences in addressing. First, mailbox addresses were allowed to have a route portion before the addr-spec when enclosed in "<" and ">". The route is simply a comma-separated list of domain names, each preceded by "@", and the list terminated by a colon. Second, CFWS were allowed between the period-separated elements of local-part and domain (i.e., dot-atom was not used). In addition, local-part is allowed to contain quoted-string in addition to just atom. Third, mailbox-list and address-list were allowed to have "null" members. That is, there could be two or more commas in such a list with nothing in between them, or commas at the beginning or end of the list. Finally, US-ASCII control characters and quoted-pairs were allowed in domain literals and are added here.

```
obs-angle-addr  =   [CFWS] "<" obs-route addr-spec ">" [CFWS]

obs-route       =   obs-domain-list ":"

obs-domain-list =   *(CFWS / ",") "@" domain
                    *("," [CFWS] ["@" domain])

obs-mbox-list   =   *([CFWS] ",") mailbox *("," [mailbox / CFWS])

obs-addr-list   =   *([CFWS] ",") address *("," [address / CFWS])

obs-group-list  =   1*([CFWS] ",") [CFWS]

obs-local-part  =   word *("." word)

obs-domain      =   atom *("." atom)

obs-dtext       =   obs-NO-WS-CTL / quoted-pair
```

When interpreting addresses, the route portion SHOULD be ignored.

4.5. Obsolete Header Fields

Syntactically, the primary difference in the obsolete field syntax is that it allows multiple occurrences of any of the fields and they may occur in any order. Also, any amount of white space is allowed before the ":" at the end of the field name.

```
obs-fields      =   *(obs-return /
                      obs-received /
                      obs-orig-date /
                      obs-from /
                      obs-sender /
                      obs-reply-to /
                      obs-to /
                      obs-cc /
                      obs-bcc /
                      obs-message-id /
                      obs-in-reply-to /
                      obs-references /
                      obs-subject /
                      obs-comments /
                      obs-keywords /
                      obs-resent-date /
                      obs-resent-from /
                      obs-resent-send /
                      obs-resent-rply /
                      obs-resent-to /
                      obs-resent-cc /
                      obs-resent-bcc /
                      obs-resent-mid /
                      obs-optional)
```

Except for destination address fields (described in section 4.5.3), the interpretation of multiple occurrences of fields is unspecified. Also, the interpretation of trace fields and resent fields that do not occur in blocks prepended to the message is unspecified as well. Unless otherwise noted in the following sections, interpretation of other fields is identical to the interpretation of their non-obsolete counterparts in section 3.

4.5.1. Obsolete Origination Date Field

```
obs-orig-date   =   "Date" *WSP ":" date-time CRLF
```

4.5.2. Obsolete Originator Fields

```
obs-from        =   "From" *WSP ":" mailbox-list CRLF

obs-sender      =   "Sender" *WSP ":" mailbox CRLF

obs-reply-to    =   "Reply-To" *WSP ":" address-list CRLF
```

4.5.3. Obsolete Destination Address Fields

```
obs-to          =   "To" *WSP ":" address-list CRLF

obs-cc          =   "Cc" *WSP ":" address-list CRLF

obs-bcc         =   "Bcc" *WSP ":"
                    (address-list / (*([CFWS] ",") [CFWS])) CRLF
```

When multiple occurrences of destination address fields occur in a message, they SHOULD be treated as if the address list in the first occurrence of the field is combined with the address lists of the subsequent occurrences by adding a comma and concatenating.

4.5.4. Obsolete Identification Fields

The obsolete "In-Reply-To:" and "References:" fields differ from the current syntax in that they allow phrase (words or quoted strings) to appear. The obsolete forms of the left and right sides of msg-id allow interspersed CFWS, making them syntactically identical to local-part and domain, respectively.

```
obs-message-id  =   "Message-ID" *WSP ":" msg-id CRLF

obs-in-reply-to =   "In-Reply-To" *WSP ":" *(phrase / msg-id) CRLF

obs-references  =   "References" *WSP ":" *(phrase / msg-id) CRLF

obs-id-left     =   local-part

obs-id-right    =   domain
```

For purposes of interpretation, the phrases in the "In-Reply-To:" and "References:" fields are ignored.

Semantically, none of the optional CFWS in the local-part and the domain is part of the obs-id-left and obs-id-right, respectively.

4.5.5. Obsolete Informational Fields

```
obs-subject     =   "Subject" *WSP ":" unstructured CRLF

obs-comments    =   "Comments" *WSP ":" unstructured CRLF

obs-keywords    =   "Keywords" *WSP ":" obs-phrase-list CRLF
```

4.5.6. Obsolete Resent Fields

The obsolete syntax adds a "Resent-Reply-To:" field, which consists of the field name, the optional comments and folding white space, the colon, and a comma separated list of addresses.

```
obs-resent-from =   "Resent-From" *WSP ":" mailbox-list CRLF

obs-resent-send =   "Resent-Sender" *WSP ":" mailbox CRLF

obs-resent-date =   "Resent-Date" *WSP ":" date-time CRLF

obs-resent-to   =   "Resent-To" *WSP ":" address-list CRLF

obs-resent-cc   =   "Resent-Cc" *WSP ":" address-list CRLF

obs-resent-bcc  =   "Resent-Bcc" *WSP ":"
                    (address-list / (*([CFWS] ",") [CFWS])) CRLF

obs-resent-mid  =   "Resent-Message-ID" *WSP ":" msg-id CRLF

obs-resent-rply =   "Resent-Reply-To" *WSP ":" address-list CRLF
```

As with other resent fields, the "Resent-Reply-To:" field is to be treated as informational only.

4.5.7. Obsolete Trace Fields

The obs-return and obs-received are again given here as template definitions, just as return and received are in section 3. Their full syntax is given in [I-D.ietf-emailcore-rfc5321bis].

```
obs-return      =   "Return-Path" *WSP ":" path CRLF

obs-received    =   "Received" *WSP ":"
                    [1*received-token / CFWS] [ ";" date-time ] CRLF
```

4.5.8. Obsolete optional fields

```
obs-optional    =   field-name *WSP ":" unstructured CRLF
```

5. Security Considerations

Care needs to be taken when displaying messages on a terminal or terminal emulator. Powerful terminals may act on escape sequences and other combinations of US-ASCII control characters with a variety of consequences. They can remap the keyboard or permit other modifications to the terminal that could lead to denial of service or even damaged data. They can trigger (sometimes programmable) answerback messages that can allow a message to cause commands to be issued on the recipient's behalf. They can also affect the operation of terminal attached devices such as printers. Message viewers may wish to strip potentially dangerous terminal escape sequences from the message prior to display. However, other escape sequences appear in messages for useful purposes (cf. [ISO.2022.1994], [RFC2045], [RFC2046], [RFC2047], [RFC2049], [BCP13]) and therefore should not be stripped indiscriminately.

Transmission of non-text objects in messages raises additional security issues. These issues are discussed in [RFC2045], [RFC2046], [RFC2047], [RFC2049], [BCP13].

The "Bcc:" (blind carbon copy) field, described in section 3.6.3, facilitates sending a message to a recipient without revealing that recipient's address to other recipients. However, use of the field itself requires care to avoid unintentional disclosures of addresses or other confidential information. For example, if using the first method described in section 3.6.3, where the "Bcc:" line is removed from the message, "Bcc:" recipients have no explicit indication that they have been sent a blind copy, except insofar as their address does not appear in the header section of a message. Because of this, a "Bcc:" recipient might send a reply to all of the listed recipients, accidentally revealing to those other recipients that the message went to the "Bcc:" recipient. Even when the second or third methods from section 3.6.3 are used, where a separate copy of the message is sent to each "Bcc:" recipient, with only the individual's address (or no address at all) appearing in the "Bcc:" field, recipient implementations still need to be careful to process replies to the message as per section 3.6.3 so as not to accidentally reveal the "Bcc:" recipient to other recipients. Similar considerations need to be given to the use of "Resent-Bcc:" field described in section 3.6.6.

6. IANA Considerations

6.1. Message Header Field Registry Changes

This document requests IANA to change the structures of the Permanent Header Field Registry and the Provisional Header Field Registry, adding an additional field called "Trace" to each entry in each registries. The value of this field is binary, and should contain either "yes" or "no" (or however IANA normally encodes such binary fields). It is only applicable if the "applicable protocol" is set "mail" (in other cases, it should be set to " ") and indicates whether the header field is to be treated as belonging to the block of trace fields as defined in section 3.6.7. This document updates [RFC3864] to add an additional field to the registration template for the Permanent Header Field Registry and the Provisional Header Field Repository as follows:

Trace: Whether or not this field belongs to the block of trace fields as defined in section 3.6.7 of [Reference to this document]. Only applicable if "Applicable Protocol" is set to "mail".

IANA is requested to update the Permanent Header Field Registry and the Provisional Header Field Registry to add this field to each entry. This field should be empty for existing registrations except for those specified in section 6.2 of this document.

6.2. Updates to the Permanent Message Header Field Registry

This document requests IANA to update the registrations that first appeared in [RFC4021] and were subsequently updated by [RFC5322]. IANA is requested to update the Permanent Message Header Field Registry with the following header fields, in accordance with the procedures set out in [RFC3864] and section 6.1.

Header field name Date
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.1)

Header field name From
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.2)

Header field name Sender

Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.2)

Header field name Reply-To
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.2)

Header field name To
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.3)

Header field name Cc
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.3)

Header field name Bcc
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.3)

Header field name Message-ID
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.4)

Header field name In-Reply-To
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.4)

Header field name References
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.4)

Header field name Subject
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.5)

Header field name Comments
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.5)

Header field name Keywords
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.5)

Header field name Resent-Date
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.6)

Header field name Resent-From
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.6)

Header field name Resent-Sender
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.6)

Header field name Resent-To
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.6)

Header field name Resent-Cc
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.6)

Header field name Resent-Bcc
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.6)

Header field name Resent-Reply-To
Applicable protocol Mail
Status obsolete
Author/Change controller IETF
Trace no
Specification document(s) This document (section 4.5.6)

Header field name Resent-Message-ID
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace no
Specification document(s) This document (section 3.6.6)

Header field name Return-Path
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace yes
Specification document(s) This document (section 3.6.7)
Related information [I-D.ietf-emailcore-rfc5321bis]

Header field name Received
Applicable protocol Mail
Status standard
Author/Change controller IETF
Trace yes

Specification document(s) This document (section 3.6.7)
Related information [I-D.ietf-emailcore-rfc5321bis]

7. References

7.1. Normative References

[ANSI.X3-4.1986]
American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[BCP14] Best Current Practice 14,
<https://www.rfc-editor.org/info/bcp14>.
At the time of writing, this BCP comprises the following:

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>.

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>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[STD13] Internet Standard 13,
<https://www.rfc-editor.org/info/std13>.
At the time of writing, this STD comprises the following:

Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[STD68] Internet Standard 68,
<https://www.rfc-editor.org/info/std68>.
At the time of writing, this STD comprises the following:

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

7.2. Informative References

[BCP13] Best Current Practice 13, <https://www.rfc-editor.org/info/bcp13>. At the time of writing, this BCP comprises the following:

Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 4289, DOI 10.17487/RFC4289, December 2005, <https://www.rfc-editor.org/info/rfc4289>.

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

[ISO.2022.1994] International Organization for Standardization, "Information technology – Character code structure and extension techniques", ISO Standard 2022, 1994.

[RFC0822] Crocker, D., "STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES", STD 11, RFC 822, DOI 10.17487/RFC0822, August 1982, <https://www.rfc-editor.org/info/rfc822>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <https://www.rfc-editor.org/info/rfc2047>.

[RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, DOI 10.17487/RFC2049, November 1996, <https://www.rfc-editor.org/info/rfc2049>.

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, DOI 10.17487/RFC2822, April 2001, <https://www.rfc-editor.org/info/rfc2822>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

[RFC4021] Klyne, G. and J. Palme, "Registration of Mail and MIME Header Fields", RFC 4021, DOI 10.17487/RFC4021, March 2005, <https://www.rfc-editor.org/info/rfc4021>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC6532] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, DOI 10.17487/RFC6532, February 2012, <https://www.rfc-editor.org/info/rfc6532>.

[I-D.ietf-emailcore-rfc5321bis] Klensin, J. C., "Simple Mail Transfer Protocol", Work in Progress, Internet-Draft, draft-ietf-emailcore-rfc5321bis-27, 26 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-emailcore-rfc5321bis-27>.

[I-D.ietf-emailcore-as] Klensin, J. C., Murchison, K., and E. Sam, "Applicability Statement for IETF Core Email Protocols", Work in Progress, Internet-Draft, draft-ietf-emailcore-as-09, 18 December 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-emailcore-as-09>.

Appendix A. Example Messages

This section presents a selection of messages. These are intended to assist in the implementation of this specification, but should not be taken as normative; that is to say, although the examples in this section were carefully reviewed, if there happens to be a conflict between these examples and the syntax described in sections 3 and 4 of this document, the syntax in those sections is to be taken as correct.

A.1. Addressing Examples

The following are examples of messages that might be sent between two individuals.

A.1.1. A Message from One Person to Another with Simple Addressing

This could be called a canonical message. It has a single author, John Doe, a single recipient, Mary Smith, a subject, the date, a message identifier, and a textual message in the body.

```
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
```

If John's secretary Michael actually sent the message, even though John was the author and replies to this message should go back to him, the sender field would be used:

```
From: John Doe <jdoe@machine.example>
Sender: Michael Jones <mjones@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
```

A.1.2. Different Types of Mailboxes

This message includes multiple addresses in the destination fields and also uses several different forms of addresses.

```
From: "Joe Q. Public" <john.q.public@example.com>
To: Mary Smith <mary@x.test>, jdoe@example.org, Who? <one@y.test>
Cc: <boss@nil.test>, "Giant; \"Big\" Box" <sysservices@example.net>
Date: Tue, 1 Jul 2003 10:52:37 +0200
Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.
```

Note that the display names for Joe Q. Public and Giant; "Big" Box needed to be enclosed in double-quotes because the former contains the period and the latter contains both semicolon and double-quote characters (the double-quote characters appearing as quoted-pair constructs). Conversely, the display name for Who? could appear without them because the question mark is legal in an atom. Notice also that jdoe@example.org and boss@nil.test have no display names associated with them at all, and jdoe@example.org uses the simpler address form without the angle brackets.

A.1.3. Group Addresses

```
From: Pete <pete@silly.example>
To: A Group:Ed Jones <e@a.test>,one@y.test,John <jdoe@one.test>;
Cc: Undisclosed recipients:;
Date: Thu, 13 Feb 1969 23:32:54 -0330
Message-ID: <testabcd.1234@silly.example>

Testing.
```

In this message, the "To:" field has a single group recipient named "A Group", which contains 3 addresses, and a "Cc:" field with an empty group recipient named Undisclosed recipients.

A.2. Reply Messages

The following is a series of three messages that make up a conversation thread between John and Mary. John first sends a message to Mary, Mary then replies to John’s message, and then John replies to Mary’s reply message.

Note especially the "Message-ID:", "References:", and "In-Reply-To:" fields in each message.

```
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
```

When sending replies, the Subject field is often retained, though prepended with "Re: " as described in section 3.6.5.

```
From: Mary Smith <mary@example.net>
To: John Doe <jdoe@machine.example>
Reply-To: "Mary Smith: Personal Account" <smith@home.example>
Subject: Re: Saying Hello
Date: Fri, 21 Nov 1997 10:01:10 -0600
Message-ID: <3456@example.net>
In-Reply-To: <1234@local.machine.example>
References: <1234@local.machine.example>

This is a reply to your hello.
```

Note the "Reply-To:" field in the above message. When John replies to Mary's message above, the reply should go to the address in the "Reply-To:" field instead of the address in the "From:" field.

```
To: "Mary Smith: Personal Account" <smith@home.example>
From: John Doe <jdoe@machine.example>
Subject: Re: Saying Hello
Date: Fri, 21 Nov 1997 11:00:00 -0600
Message-ID: <abcd.1234@local.machine.test>
In-Reply-To: <3456@example.net>
References: <1234@local.machine.example> <3456@example.net>

This is a reply to your reply.
```

A.3. Resent Messages

Start with the message that has been used as an example several times:

```
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
```

Say that Mary, upon receiving this message, wishes to send a copy of the message to Jane such that (a) the message would appear to have come straight from John; (b) if Jane replies to the message, the reply should go back to John; and (c) all of the original information, like the date the message was originally sent to Mary, the message identifier, and the original addressee, is preserved. In this case, resent fields are prepended to the message:

```
Resent-From: Mary Smith <mary@example.net>
Resent-To: Jane Brown <j-brown@other.example>
Resent-Date: Mon, 24 Nov 1997 14:22:01 -0800
Resent-Message-ID: <78910@example.net>
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
```

If Jane, in turn, wished to resend this message to another person, she would prepend her own set of resent header fields to the above and send that. (Note that for brevity, trace fields are not shown.)

A.4. Messages with Trace Fields

As messages are sent through the transport system as described in [I-D.ietf-emailcore-rfc5321bis], trace fields are prepended to the message. The following is an example of what those trace fields might look like. Note that there is some folding white space in the first one since these lines can be long.

```
Received: from x.y.test
   by example.net
   via TCP
   with ESMTP
   id ABC12345
   for <mary@example.net>;  21 Nov 1997 10:05:43 -0600
Received: from node.example by x.y.test; 21 Nov 1997 10:01:22 -0600
From: John Doe <jdoe@node.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.node.example>

This is a message just to say hello.
So, "Hello".
```

A.5. White Space, Comments, and Other Oddities

White space, including folding white space, and comments can be inserted between many of the tokens of fields. Taking the example from A.1.3, white space and comments can be inserted into all of the fields.

```
From: Pete(A nice \) chap) <pete@silly.test(his host is silly)>
To:A Group(Some people)
     :Ed Jones <e@a.test(.host of Ed)>,
         one@y.test,
  John <jdoe@one.test> (my dear friend); (the end of the group)
Cc:(Empty list)(start)Hidden recipients  :(nobody(that I know))  ;
Date: Thu,
      13
        Feb
          1969
      23:32
                -0330 (Newfoundland Time)
Message-ID:              <testabcd.1234@silly.test>

Testing.
```

The above example is aesthetically displeasing, but perfectly legal. Note particularly (1) the comments in the "From:" field (including one that has a ")" character appearing as part of a quoted-pair); (2) the white space absent after the ":" in the "To:" field as well as the comment and folding white space after the group name, the special character (".") in the comment in Ed Jones's address, and the folding white space before and after "one@y.test,"; (3) the multiple and nested comments in the "Cc:" field as well as the comment immediately following the ":" after "Cc"; (4) the folding white space (but no

comments except at the end) and the missing seconds in the time of the date field; and (5) the white space before (but not within) the identifier in the "Message-ID:" field.

A.6. Obsoleted Forms

The following are examples of obsolete (that is, the "MUST NOT generate") syntactic elements described in section 4 of this document.

A.6.1. Obsolete Addressing

Note in the example below the lack of quotes around Joe Q. Public, the route that appears in the address for Mary Smith, the two commas that appear in the "To:" field, and the spaces that appear around the "." in the jdoe address.

```
From: Joe Q. Public <john.q.public@example.com>
To: Mary Smith <@node.test:mary@example.net>, , jdoe@one  . test
Date: Tue, 1 Jul 2003 10:52:37 +0200
Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.
```

A.6.2. Obsolete Dates

The following message uses an obsolete date format, including a non-numeric time zone and a two digit year. Note that although the day-of-week is missing, that is not specific to the obsolete syntax; it is optional in the current syntax as well.

```
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Subject: Saying Hello
Date: 21 Nov 97 09:55:06 GMT
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
```

A.6.3. Obsolete White Space and Comments

White space and comments can appear between many more elements than in the current syntax. Also, folding lines that are made up entirely of white space are legal.

```
From  : John Doe <jdoe@machine(comment).  example>
To    : Mary Smith
__
          <mary@example.net>
Subject     : Saying Hello
Date  : Fri, 21 Nov 1997 09(comment):   55  :  06 -0600
Message-ID  : <1234   @   local(blah)  .machine .example>

This is a message just to say hello.
So, "Hello".
```

Note especially the second line of the "To:" field. It starts with two space characters. (Note that "__" represents blank spaces.) Therefore, it is considered part of the folding, as described in section 4.2. Also, the comments and white space throughout addresses, dates, and message identifiers are all part of the obsolete syntax.

Appendix B. Differences from Earlier Specifications

This appendix contains a list of changes that have been made in the Internet Message Format from earlier specifications, specifically [RFC0822], [RFC1123], [RFC2822], and [RFC5322]. Items marked with an asterisk (*) below are items which appear in section 4 of this document and therefore can no longer be generated.

The following are the changes made from [RFC0822] and [RFC1123] to [RFC2822]:

1. Period allowed in obsolete form of phrase.
2. ABNF moved out of document, now in [STD68].
3. Four or more digits allowed for year.
4. Header field ordering (and lack thereof) made explicit.
5. Encrypted header field removed.
6. Specifically allow and give meaning to "-0000" time zone.
7. Folding white space is not allowed between every token.
8. Requirement for destinations removed.
9. Forwarding and resending redefined.
10. Extension header fields no longer specifically called out.
11. ASCII 0 (null) removed.*
12. Folding continuation lines cannot contain only white space.*
13. Free insertion of comments not allowed in date.*
14. Non-numeric time zones not allowed.*
15. Two digit years not allowed.*
16. Three digit years interpreted, but not allowed for generation.*
17. Routes in addresses not allowed.*
18. CFWS within local-parts and domains not allowed.*
19. Empty members of address lists not allowed.*

20. Folding white space between field name and colon not allowed.*
21. Comments between field name and colon not allowed.
22. Tightened syntax of in-reply-to and references.*
23. CFWS within msg-id not allowed.*
24. Tightened semantics of resent fields as informational only.
25. Resent-Reply-To not allowed.*
26. No multiple occurrences of fields (except resent and received).*
27. Free CR and LF not allowed.*
28. Line length limits specified.
29. Bcc more clearly specified.

The following are changes from [RFC2822] to [RFC5322]:

1. Assorted typographical/grammatical errors fixed and clarifications made.
2. Changed "standard" to "document" or "specification" throughout.
3. Made distinction between "header field" and "header section".
4. Removed NO-WS-CTL from ctext, qtext, dtext, and unstructured.*
5. Moved discussion of specials to the "Atom" section. Moved text to "Overall message syntax" section.
6. Simplified CFWS syntax.
7. Fixed unstructured syntax (erratum 373 (https://www.rfc-editor.org/errata/eid373)).
8. Changed date and time syntax to deal with white space in obsolete date syntax.
9. Removed quoted-pair from domain literals and message identifiers.*
10. Clarified that other specifications limit domain syntax.
11. Simplified "Bcc:" and "Resent-Bcc:" syntax.
12. Allowed optional-field to appear within trace information.
13. Removed no-fold-quote from msg-id. Clarified syntax limitations.
14. Generalized "Received:" syntax to fix bugs and move definition out of this document.
15. Simplified obs-qp. Fixed and simplified obs-utext (which now only appears in the obsolete syntax). Removed obs-text and obs-char, adding obs-body.
16. Fixed obsolete date syntax to allow for more (or less) comments and white space.
17. Fixed all obsolete list syntax (obs-domain-list, obs-mbox-list, obs-addr-list, obs-phrase-list, and the newly added obs-group-list).
18. Fixed obs-reply-to syntax.
19. Fixed obs-bcc and obs-resent-bcc to allow empty lists.
20. Removed obs-path.

The following are changes from [RFC5322].

1. Clarified addr-spec description (erratum 1766 (https://www.rfc-editor.org/errata/eid1766)).
2. Fixed obs-unstruct to be more limited (erratum 1905 (https://www.rfc-editor.org/errata/eid1905)).*
3. Simplified obs-body (erratum 1906 (https://www.rfc-editor.org/errata/eid1906)).*
4. Fixed obs-FWS to allow for a leading CRLF (erratum 1908 (https://www.rfc-editor.org/errata/eid1908)).*
5. Fixed comments within addresses in A.5 (errata 2515 (https://www.rfc-editor.org/errata/eid2515) and 2579 (https://www.rfc-editor.org/errata/eid2579)).
6. Fixed time zone description (erratum 2726 (https://www.rfc-editor.org/errata/eid2726)).
7. Removed inappropriate uses of "sent" in 3.6.3, 3.6.6, and 5 (erratum 3048 (https://www.rfc-editor.org/errata/eid3048)).
8. Allowed for CFWS in otherwise empty list of "Received:" field tokens (erratum 3979 (https://www.rfc-editor.org/errata/eid3979)).
9. Clarified that "printable" includes space, and replaced "printable" with "VCHAR" in ABNF comments to clarify that it doesn’t include the space character (erratum 4692 (https://www.rfc-editor.org/errata/eid4692)).
10. Clarified midnight in time-of-day (erratum 5905 (https://www.rfc-editor.org/errata/eid5905)).
11. Allowed for date-time in obs-received (erratum 5867 (https://www.rfc-editor.org/errata/eid5867)).*
12. Separated out "msg-id-internal" in "msg-id" (2822bis list discussion (https://mailarchive.ietf.org/arch/browse/ietf-822/?gbt=1&index=g2ocdJIKBHBF8CrC3HxBcLR3-M0)).
13. Updated references to STD 13, STD 68, BCP 13, and BCP 14, and reference for leap seconds to RFC 3339.
14. Fixed typo in daylight saving time in description of obs-zone.*
15. Added comment to field-name ABNF to remind that length can’t be greater than 77 (erratum 5918 (https://www.rfc-editor.org/errata/eid5918)).
16. Clarified description in 4.5.6 as "trace information"".
17. Explained the use of the term "obsolete" in Section 1.2.3.
18. Updated syntactic and semantic descriptions of trace in 3.6.7 to allow other fields that are treated as trace, and allow return-path without any received. Moved optional-field syntax into this section and out of the top portion of 3.6 to accommodate this.
19. Updated Header Field Registries to include a "Trace" column.
20. Added optional CFWS around obs-zone (erratum 6639 (https://www.rfc-editor.org/errata/eid6639))
21. Changed "Note:" paragraphs with normative text in sections 2.2.3, 3.4, and 3.6.6 to normal paragraphs and removed normative text from "Note:" in section in 4.

22. Rewrote "Bcc:" section (Section 3.6.3) to clarify and correct error that allowed blank "Bcc:" line to go to "To:" and "Cc:" recipients
23. Changed Security Considerations (Section 5) to account for changes to "Bcc:" clarifications and added note about "Reent-Bcc:"
24. Indented all examples to look clearer in the text version. Changed the "To:" field body in A.1.3, A.5, and A.6.1 to shorten the lines and make them consistent.

// This last part to be removed before publication.

There are also 2 errata that were "Held For Document Update" that have not been addressed:

1. Erratum 2950 (https://www.rfc-editor.org/errata/eid2950): As per ticket #39 (https://trac.ietf.org/trac/emailcore/ticket/39), there is no need to change the resent fields from "*" to "1*" in 3.6 as it doesn't really affect the syntax.
2. Erratum 3135 (https://www.rfc-editor.org/errata/eid3135): As per ticket #35 (https://trac.ietf.org/trac/emailcore/ticket/35), discussion of empty quoted-string will appear in https://datatracker.ietf.org/doc/draft-ietf-emailcore-as/

Appendix C. Acknowledgements

Many people contributed to this document. They included participants in and chairs of the Detailed Revision and Update of Messaging Standards (DRUMS), Yet Another Mail (YAM), and Revision of Core Email Specifications (EMAILCORE) Working Groups of the Internet Engineering Task Force (IETF), the Area Directors of the IETF, reporters of errata on earlier versions of this document, and people who simply sent their comments in via email. The editor is deeply indebted to them all and thanks them sincerely. (While the editor wishes to thank them all by name as was done in the past, the list has gotten so long to make including it here untenable. But the thanks is no less heartfelt.)

Author's Address

Peter W. Resnick (editor)
Episteme Technology Consulting LLC
503 West Indiana Avenue
Urbana, IL 61801-4941
United States of America
Phone: +1 217 337 1905
Email: resnick@episteme.net

URI: https://www.episteme.net/

HTTP
Internet-Draft
Obsoletes: 8941 (if approved)
Intended status: Standards Track
Expires: 23 October 2024

M. Nottingham
Cloudflare
P.-H. Kamp
The Varnish Cache Project

21 April 2024

Structured Field Values for HTTP
draft-ietf-httpbis-sfbis-06

Abstract

This document describes a set of data types and associated algorithms that are intended to make it easier and safer to define and handle HTTP header and trailer fields, known as "Structured Fields", "Structured Headers", or "Structured Trailers". It is intended for use by specifications of new HTTP fields that wish to use a common syntax that is more restrictive than traditional HTTP field values.

This document obsoletes RFC 8941.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpbis-sfbis/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group information can be found at https://httpwg.org/.

Source for this draft and an issue tracker can be found at https://github.com/httpwg/http-extensions/labels/header-structure.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Specifying the syntax of new HTTP header (and trailer) fields is an onerous task; even with the guidance in Section 16.3.2 of [HTTP], there are many decisions -- and pitfalls -- for a prospective HTTP field author.

Once a field is defined, bespoke parsers and serializers often need to be written, because each field value has a slightly different handling of what looks like common syntax.

This document introduces a set of common data structures for use in definitions of new HTTP field values to address these problems. In particular, it defines a generic, abstract model for them, along with a concrete serialization for expressing that model in HTTP [HTTP] header and trailer fields.

An HTTP field that is defined as a "Structured Header" or "Structured Trailer" (if the field can be either, it is a "Structured Field") uses the types defined in this specification to define its syntax and basic handling rules, thereby simplifying both its definition by specification writers and handling by implementations.

Additionally, future versions of HTTP can define alternative serializations of the abstract model of these structures, allowing fields that use that model to be transmitted more efficiently without being redefined.

Note that it is not a goal of this document to redefine the syntax of existing HTTP fields; the mechanisms described herein are only intended to be used with fields that explicitly opt into them.

Section 2 describes how to specify a Structured Field.

Section 3 defines a number of abstract data types that can be used in Structured Fields.

Those abstract types can be serialized into and parsed from HTTP field values using the algorithms described in Section 4.

1.1. Intentionally Strict Processing

This specification intentionally defines strict parsing and serialization behaviors using step-by-step algorithms; the only error handling defined is to fail the entire operation altogether.

It is designed to encourage faithful implementation and good interoperability. Therefore, an implementation that tried to be helpful by being more tolerant of input would make interoperability worse, since that would create pressure on other implementations to implement similar (but likely subtly different) workarounds.

In other words, strict processing is an intentional feature of this specification; it allows non-conformant input to be discovered and corrected by the producer early and avoids both interoperability and security issues that might otherwise result.

Note that as a result of this strictness, if a field is appended to by multiple parties (e.g., intermediaries or different components in the sender), an error in one party's value is likely to cause the entire field value to fail parsing.

1.2. Notational Conventions

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

This document uses the VCHAR, SP, DIGIT, ALPHA, and DQUOTE rules from [RFC5234] to specify characters and/or their corresponding ASCII bytes, depending on context. It uses the tchar and OWS rules from [HTTP] for the same purpose.

This document uses algorithms to specify parsing and serialization behaviors. When parsing from HTTP fields, implementations MUST have behavior that is indistinguishable from following the algorithms.

For serialization to HTTP fields, the algorithms define the recommended way to produce them. Implementations MAY vary from the specified behavior so long as the output is still correctly handled by the parsing algorithm described in Section 4.2.

2. Defining New Structured Fields

To specify an HTTP field as a Structured Field, its authors need to:

* Normatively reference this specification. Recipients and generators of the field need to know that the requirements of this document are in effect.

* Identify whether the field is a Structured Header (i.e., it can only be used in the header section -- the common case), a Structured Trailer (only in the trailer section), or a Structured Field (both).

* Specify the type of the field value; either List (Section 3.1), Dictionary (Section 3.2), or Item (Section 3.3).

* Define the semantics of the field value.

* Specify any additional constraints upon the field value, as well as the consequences when those constraints are violated.

Typically, this means that a field definition will specify the top-level type -- List, Dictionary, or Item -- and then define its allowable types and constraints upon them. For example, a header defined as a List might have all Integer members, or a mix of types; a header defined as an Item might allow only Strings, and additionally only strings beginning with the letter "Q", or strings in lowercase. Likewise, Inner Lists (Section 3.1.1) are only valid when a field definition explicitly allows them.

Fields that use the Display String type are advised to carefully specify their allowable unicode code points; for example, specifying the use of a profile from [PRECIS].

Field definitions can only use this specification for the entire field value, not a portion thereof.

Specifications can refer to a field name as a "structured header name", "structured trailer name", or "structured field name" as appropriate. Likewise, they can refer its field value as a "structured header value", "structured trailer value", or "structured field value" as necessary.

This specification defines minimums for the length or number of various structures supported by implementations. It does not specify maximum sizes in most cases, but authors should be aware that HTTP implementations do impose various limits on the size of individual fields, the total number of fields, and/or the size of the entire header or trailer section.

2.1. Example

A fictitious Foo-Example header field might be specified as:

> 42. Foo-Example Header Field
>
> The Foo-Example HTTP header field conveys information about how much Foo the message has.
>
> Foo-Example is an Item Structured Header Field [RFCnnnn]. Its value MUST be an Integer (Section 3.3.1 of [RFCnnnn]).
>
> Its value indicates the amount of Foo in the message, and it MUST be between 0 and 10, inclusive; other values MUST cause the entire header field to be ignored.
>
> The following parameter is defined:

> * A parameter whose key is "foourl", and whose value is a String (Section 3.3.3 of [RFCnnnn]), conveying the Foo URL for the message. See below for processing requirements.
>
> "foourl" contains a URI-reference (Section 4.1 of [RFC3986]). If its value is not a valid URI-reference, the entire header field MUST be ignored. If its value is a relative reference (Section 4.2 of [RFC3986]), it MUST be resolved (Section 5 of [RFC3986]) before being used.
>
> For example:
>
> ```
> Foo-Example: 2; foourl="https://foo.example.com/"
> ```

2.2. Error Handling

When parsing fails, the entire field is ignored (see Section 4.2). Field definitions cannot override this, because doing so would preclude handling by generic software; they can only add additional constraints (for example, on the numeric range of Integers and Decimals, the format of Strings and Tokens, the types allowed in a Dictionary's values, or the number of Items in a List).

When field-specific constraints are violated, the entire field is also ignored, unless the field definition defines other handling requirements. For example, if a header field is defined as an Item and required to be an Integer, but a String is received, it should be ignored unless that field's definition explicitly specifies otherwise.

2.3. Preserving Extensibility

Structured Fields are designed to be extensible, because experience has shown that even when it is not foreseen, it is often necessary to modify and add to the allowable syntax and semantics of a field in a controlled fashion.

Both Items and Inner Lists allow Parameters as an extensibility mechanism; this means that their values can later be extended to accommodate more information, if need be. To preserve forward compatibility, field specifications are discouraged from defining the presence of an unrecognized parameter as an error condition.

Field specifications are required to be either an Item, List, or Dictionary to preserve extensibility. Fields that erroneously defined as another type (e.g., Integer) are assumed to be Items (i.e., they allow Parameters).

To further assure that this extensibility is available in the future, and to encourage consumers to use a complete parser implementation, a field definition can specify that "grease" parameters be added by senders. A specification could stipulate that all parameters that fit a defined pattern are reserved for this use and then encourage them to be sent on some portion of requests. This helps to discourage recipients from writing a parser that does not account for Parameters.

Specifications that use Dictionaries can also allow for forward compatibility by requiring that the presence of -- as well as value and type associated with -- unknown keys be ignored. Subsequent specifications can then add additional keys, specifying constraints on them as appropriate.

An extension to a Structured Field can then require that an entire field value be ignored by a recipient that understands the extension if constraints on the value it defines are not met.

2.4. Using New Structured Types in Extensions

Because a field definition needs to reference a specific RFC for Structured Fields, the types available for use in its value are limited to those defined in that RFC. For example, a field whose definition references this document can have a value that uses the Date type (Section 3.3.7), whereas a field whose definition references RFC 8941 cannot, because it will be treated as invalid (and therefore discarded) by implementations of that specification.

This limitation also applies to future extensions to a field; for example, a field that is defined with reference to RFC 8941 cannot use the Date type, because some recipients might still be using an RFC 8941 parser to process it.

However, this document is designed to be backwards-compatible with RFC 8941; a parser that implements the requirements here can also parse valid Structured Fields whose definitions reference RFC 8941.

Upgrading a Structured Fields implementation to support a newer revision of the specification (such as this document) brings the possibility that some field values that were invalid according to the earlier RFC might become valid when processed.

For example, field instance might contain a syntactically valid Date (Section 3.3.7), even though that field's definition does not accommodate Dates. An RFC8941 implementation would fail parsing such a field instance, because they are not defined in that specification. If that implementation were upgraded to this specification, parsing

would now succeed. In some cases, the resulting Date value will be rejected by field-specific logic, but values in fields that are otherwise ignored (such as extension parameters) might not be detected and the field might subsequently be accepted and processed.

3. Structured Data Types

This section provides an overview of the abstract types that Structured Fields use, and gives a brief description and examples of how each of those types are serialized into textual HTTP fields. Section 4 specifies the details of how they are parsed from and serialized into textual HTTP fields.

In summary:

* There are three top-level types that an HTTP field can be defined as: Lists, Dictionaries, and Items.

* Lists and Dictionaries are containers; their members can be Items or Inner Lists (which are themselves arrays of Items).

* Both Items and Inner Lists can be Parameterized with key/value pairs.

3.1. Lists

Lists are arrays of zero or more members, each of which can be an Item (Section 3.3) or an Inner List (Section 3.1.1), both of which can be Parameterized (Section 3.1.2).

An empty List is denoted by not serializing the field at all. This implies that fields defined as Lists have a default empty value.

When serialized as a textual HTTP field, each member is separated by a comma and optional whitespace. For example, a field whose value is defined as a List of Tokens could look like:

Example-List: sugar, tea, rum

Note that Lists can have their members split across multiple lines of the same header or trailer section, as per Section 5.3 of [HTTP]; for example, the following are equivalent:

Example-List: sugar, tea, rum

and

```
Example-List: sugar, tea
Example-List: rum
```

However, individual members of a List cannot be safely split between lines; see Section 4.2 for details.

Parsers MUST support Lists containing at least 1024 members. Field specifications can constrain the types and cardinality of individual List values as they require.

3.1.1. Inner Lists

An Inner List is an array of zero or more Items (Section 3.3). Both the individual Items and the Inner List itself can be Parameterized (Section 3.1.2).

When serialized in a textual HTTP field, Inner Lists are denoted by surrounding parenthesis, and their values are delimited by one or more spaces. A field whose value is defined as a List of Inner Lists of Strings could look like:

```
Example-List: ("foo" "bar"), ("baz"), ("bat" "one"), ()
```

Note that the last member in this example is an empty Inner List.

A header field whose value is defined as a List of Inner Lists with Parameters at both levels could look like:

```
Example-List: ("foo"; a=1;b=2);lvl=5, ("bar" "baz");lvl=1
```

Parsers MUST support Inner Lists containing at least 256 members. Field specifications can constrain the types and cardinality of individual Inner List members as they require.

3.1.2. Parameters

Parameters are an ordered map of key-value pairs that are associated with an Item (Section 3.3) or Inner List (Section 3.1.1). The keys are unique within the scope of the Parameters they occur within, and the values are bare items (i.e., they themselves cannot be parameterized; see Section 3.3).

Implementations MUST provide access to Parameters both by index and by key. Specifications MAY use either means of accessing them.

Note that parameters are ordered, and parameter keys cannot contain uppercase letters.

When serialized in a textual HTTP field, a Parameter is separated from its Item or Inner List and other Parameters by a semicolon. For example:

```
Example-List: abc;a=1;b=2; cde_456, (ghi;jk=4 l);q="9";r=w
```

Parameters whose value is Boolean (see Section 3.3.6) true MUST omit that value when serialized. For example, the "a" parameter here is true, while the "b" parameter is false:

```
Example-Integer: 1; a; b=?0
```

Note that this requirement is only on serialization; parsers are still required to correctly handle the true value when it appears in a parameter.

Parsers MUST support at least 256 parameters on an Item or Inner List, and support parameter keys with at least 64 characters. Field specifications can constrain the order of individual parameters, as well as their values' types as required.

3.2. Dictionaries

Dictionaries are ordered maps of key-value pairs, where the keys are short textual strings and the values are Items (Section 3.3) or arrays of Items, both of which can be Parameterized (Section 3.1.2). There can be zero or more members, and their keys are unique in the scope of the Dictionary they occur within.

Implementations MUST provide access to Dictionaries both by index and by key. Specifications MAY use either means of accessing the members.

As with Lists, an empty Dictionary is represented by omitting the entire field. This implies that fields defined as Dictionaries have a default empty value.

Typically, a field specification will define the semantics of Dictionaries by specifying the allowed type(s) for individual members by their keys, as well as whether their presence is required or optional. Recipients MUST ignore members whose keys are undefined or unknown, unless the field's specification specifically disallows them.

When serialized as a textual HTTP field, Members are ordered as serialized and separated by a comma with optional whitespace. Member keys cannot contain uppercase characters. Keys and values are separated by "=" (without whitespace). For example:

```
Example-Dict: en="Applepie", da=:w4ZibGV0w6ZydGU=:
```

Note that in this example, the final "=" is due to the inclusion of a Byte Sequence; see Section 3.3.5.

Members whose value is Boolean (see Section 3.3.6) true MUST omit that value when serialized. For example, here both "b" and "c" are true:

```
Example-Dict: a=?0, b, c; foo=bar
```

Note that this requirement is only on serialization; parsers are still required to correctly handle the true Boolean value when it appears in Dictionary values.

A Dictionary with a member whose value is an Inner List of Tokens:

```
Example-Dict: rating=1.5, feelings=(joy sadness)
```

A Dictionary with a mix of Items and Inner Lists, some with parameters:

```
Example-Dict: a=(1 2), b=3, c=4;aa=bb, d=(5 6);valid
```

Note that Dictionaries can have their members split across multiple lines of the same header or trailer section; for example, the following are equivalent:

```
Example-Dict: foo=1, bar=2
```

and

```
Example-Dict: foo=1
Example-Dict: bar=2
```

However, individual members of a Dictionary cannot be safely split between lines; see Section 4.2 for details.

Parsers MUST support Dictionaries containing at least 1024 key/value pairs and keys with at least 64 characters. Field specifications can constrain the order of individual Dictionary members, as well as their values' types as required.

3.3. Items

An Item can be an Integer (Section 3.3.1), a Decimal (Section 3.3.2), a String (Section 3.3.3), a Token (Section 3.3.4), a Byte Sequence (Section 3.3.5), a Boolean (Section 3.3.6), or a Date (Section 3.3.7). It can have associated parameters (Section 3.1.2).

For example, a header field that is defined to be an Item that is an Integer might look like:

```
Example-Integer: 5
```

or with parameters:

```
Example-Integer: 5; foo=bar
```

3.3.1. Integers

Integers have a range of -999,999,999,999,999 to 999,999,999,999,999 inclusive (i.e., up to fifteen digits, signed), for IEEE 754 compatibility [IEEE754].

For example:

```
Example-Integer: 42
```

Integers larger than 15 digits can be supported in a variety of ways; for example, by using a String (Section 3.3.3), a Byte Sequence (Section 3.3.5), or a parameter on an Integer that acts as a scaling factor.

While it is possible to serialize Integers with leading zeros (e.g., "0002", "-01") and signed zero ("-0"), these distinctions may not be preserved by implementations.

Note that commas in Integers are used in this section's prose only for readability; they are not valid in the wire format.

3.3.2. Decimals

Decimals are numbers with an integer and a fractional component. The integer component has at most 12 digits; the fractional component has at most three digits.

For example, a header whose value is defined as a Decimal could look like:

```
Example-Decimal: 4.5
```

While it is possible to serialize Decimals with leading zeros (e.g., "0002.5", "-01.334"), trailing zeros (e.g., "5.230", "-0.40"), and signed zero (e.g., "-0.0"), these distinctions may not be preserved by implementations.

Note that the serialization algorithm (Section 4.1.5) rounds input with more than three digits of precision in the fractional component. If an alternative rounding strategy is desired, this should be specified by the field definition to occur before serialization.

3.3.3. Strings

Strings are zero or more printable ASCII [RFC0020] characters (i.e., the range %x20 to %x7E). Note that this excludes tabs, newlines, carriage returns, etc.

Non-ASCII characters are not directly supported in Strings, because they cause a number of interoperability issues, and -- with few exceptions -- field values do not require them.

When it is necessary for a field value to convey non-ASCII content, a Display String (Section 3.3.8) can be specified.

When serialized in a textual HTTP field, Strings are delimited with double quotes, using a backslash ("\") to escape double quotes and backslashes. For example:

```
Example-String: "hello world"
```

Note that Strings only use DQUOTE as a delimiter; single quotes do not delimit Strings. Furthermore, only DQUOTE and "\" can be escaped; other characters after "\" MUST cause parsing to fail.

Parsers MUST support Strings (after any decoding) with at least 1024 characters.

3.3.4. Tokens

Tokens are short textual words that begin with an alphabetic character or "*", followed by zero to many token characters, which are the same as those allowed by the "token" ABNF rule defined in [HTTP], plus the ":" and "/" characters.

For example:

```
Example-Token: foo123/456
```

Parsers MUST support Tokens with at least 512 characters.

Note that Tokens are defined largely for compatibility with the data model of existing HTTP fields, and may require additional steps to use in some implementations. As a result, new fields are encouraged to use Strings.

3.3.5. Byte Sequences

Byte Sequences can be conveyed in Structured Fields.

When serialized in a textual HTTP field, a Byte Sequence is delimited with colons and encoded using base64 ([RFC4648], Section 4). For example:

Example-ByteSequence: :cHJldGVuZCB0aGlzIGlzIGJpbmFyeSBjb250ZW50Lg==:

Parsers MUST support Byte Sequences with at least 16384 octets after decoding.

3.3.6. Booleans

Boolean values can be conveyed in Structured Fields.

When serialized in a textual HTTP field, a Boolean is indicated with a leading "?" character followed by a "1" for a true value or "0" for false. For example:

Example-Boolean: ?1

Note that in Dictionary (Section 3.2) and Parameter (Section 3.1.2) values, Boolean true is indicated by omitting the value.

3.3.7. Dates

Date values can be conveyed in Structured Fields.

Dates have a data model that is similar to Integers, representing a (possibly negative) delta in seconds from 1970-01-01T00:00:00Z, excluding leap seconds. Accordingly, their serialization in textual HTTP fields is similar to that of Integers, distinguished from them with a leading "@".

For example:

Example-Date: @1659578233

Parsers MUST support Dates whose values include all days in years 1 to 9999 (i.e., -62,135,596,800 to 253,402,214,400 delta seconds from 1970-01-01T00:00:00Z).

3.3.8. Display Strings

Display Strings are similar to Strings, in that they consist of zero or more characters, but they allow Unicode scalar values (i.e., all Unicode code points except for surrogates), unlike Strings.

Display Strings are intended for use in cases where a value is displayed to end users, and therefore may need to carry non-ASCII content. It is NOT RECOMMENDED that they be used in situations where a String (Section 3.3.3) or Token (Section 3.3.4) would be adequate, because Unicode has processing considerations (e.g., normalization) and security considerations (e.g., homograph attacks) that make it more difficult to handle correctly.

Note that Display Strings do not indicate the language used in the value; that can be done separately if necessary (e.g., with a parameter).

In textual HTTP fields, Display Strings are represented in a manner similar to Strings, except that non-ASCII characters are percent-encoded; there is a leading "%" to distinguish them from Strings.

For example:

```
Example-DisplayString: %"This is intended for display to %c3%bcsers."
```

See Section 6 for additional security considerations when handling Display Strings.

4. Working with Structured Fields in HTTP

This section defines how to serialize and parse the abstract types defined by Section 3 into textual HTTP field values and other encodings compatible with them (e.g., in HTTP/2 [HTTP/2] before compression with HPACK [HPACK]).

4.1. Serializing Structured Fields

Given a structure defined in this specification, return an ASCII string suitable for use in an HTTP field value.

1. If the structure is a Dictionary or List and its value is empty (i.e., it has no members), do not serialize the field at all (i.e., omit both the field-name and field-value).

2. If the structure is a List, let output_string be the result of running Serializing a List (Section 4.1.1) with the structure.

3. Else, if the structure is a Dictionary, let output_string be the result of running Serializing a Dictionary (Section 4.1.2) with the structure.

4. Else, if the structure is an Item, let output_string be the result of running Serializing an Item (Section 4.1.3) with the structure.

5. Else, fail serialization.

6. Return output_string converted into an array of bytes, using ASCII encoding [RFC0020].

4.1.1. Serializing a List

Given an array of (member_value, parameters) tuples as input_list, return an ASCII string suitable for use in an HTTP field value.

1. Let output be an empty string.

2. For each (member_value, parameters) of input_list:

    1. If member_value is an array, append the result of running Serializing an Inner List (Section 4.1.1.1) with (member_value, parameters) to output.

    2. Otherwise, append the result of running Serializing an Item (Section 4.1.3) with (member_value, parameters) to output.

    3. If more member_values remain in input_list:

        1. Append "," to output.

        2. Append a single SP to output.

3. Return output.

4.1.1.1. Serializing an Inner List

Given an array of (member_value, parameters) tuples as inner_list, and parameters as list_parameters, return an ASCII string suitable for use in an HTTP field value.

1. Let output be the string "(".

2. For each (member_value, parameters) of inner_list:

1. Append the result of running Serializing an Item (Section 4.1.3) with (member_value, parameters) to output.

2. If more values remain in inner_list, append a single SP to output.

3. Append ")" to output.

4. Append the result of running Serializing Parameters (Section 4.1.1.2) with list_parameters to output.

5. Return output.

4.1.1.2. Serializing Parameters

Given an ordered Dictionary as input_parameters (each member having a param_key and a param_value), return an ASCII string suitable for use in an HTTP field value.

1. Let output be an empty string.

2. For each param_key with a value of param_value in input_parameters:

    1. Append ";" to output.

    2. Append the result of running Serializing a Key (Section 4.1.1.3) with param_key to output.

    3. If param_value is not Boolean true:

        1. Append "=" to output.

        2. Append the result of running Serializing a bare Item (Section 4.1.3.1) with param_value to output.

3. Return output.

4.1.1.3. Serializing a Key

Given a key as input_key, return an ASCII string suitable for use in an HTTP field value.

1. Convert input_key into a sequence of ASCII characters; if conversion fails, fail serialization.

2. If input_key contains characters not in lcalpha, DIGIT, "_", "-", ".", or "*", fail serialization.

3. If the first character of input_key is not lcalpha or "*", fail serialization.

4. Let output be an empty string.

5. Append input_key to output.

6. Return output.

4.1.2. Serializing a Dictionary

Given an ordered Dictionary as input_dictionary (each member having a member_key and a tuple value of (member_value, parameters)), return an ASCII string suitable for use in an HTTP field value.

1. Let output be an empty string.

2. For each member_key with a value of (member_value, parameters) in input_dictionary:

    1. Append the result of running Serializing a Key (Section 4.1.1.3) with member's member_key to output.

    2. If member_value is Boolean true:

        1. Append the result of running Serializing Parameters (Section 4.1.1.2) with parameters to output.

    3. Otherwise:

        1. Append "=" to output.

        2. If member_value is an array, append the result of running Serializing an Inner List (Section 4.1.1.1) with (member_value, parameters) to output.

        3. Otherwise, append the result of running Serializing an Item (Section 4.1.3) with (member_value, parameters) to output.

    4. If more members remain in input_dictionary:

        1. Append "," to output.

        2. Append a single SP to output.

3. Return output.

4.1.3. Serializing an Item

Given an Item as bare_item and Parameters as item_parameters, return an ASCII string suitable for use in an HTTP field value.

1. Let output be an empty string.
2. Append the result of running Serializing a Bare Item (Section 4.1.3.1) with bare_item to output.
3. Append the result of running Serializing Parameters (Section 4.1.1.2) with item_parameters to output.
4. Return output.

4.1.3.1. Serializing a Bare Item

Given an Item as input_item, return an ASCII string suitable for use in an HTTP field value.

1. If input_item is an Integer, return the result of running Serializing an Integer (Section 4.1.4) with input_item.
2. If input_item is a Decimal, return the result of running Serializing a Decimal (Section 4.1.5) with input_item.
3. If input_item is a String, return the result of running Serializing a String (Section 4.1.6) with input_item.
4. If input_item is a Token, return the result of running Serializing a Token (Section 4.1.7) with input_item.
5. If input_item is a Byte Sequence, return the result of running Serializing a Byte Sequence (Section 4.1.8) with input_item.
6. If input_item is a Boolean, return the result of running Serializing a Boolean (Section 4.1.9) with input_item.
7. If input_item is a Date, return the result of running Serializing a Date (Section 4.1.10) with input_item.
8. If input_item is a Display String, return the result of running Serializing a Display String (Section 4.1.11) with input_item.
9. Otherwise, fail serialization.

4.1.4. Serializing an Integer

Given an Integer as input_integer, return an ASCII string suitable for use in an HTTP field value.

1. If input_integer is not an integer in the range of -999,999,999,999,999 to 999,999,999,999,999 inclusive, fail serialization.

2. Let output be an empty string.

3. If input_integer is less than (but not equal to) 0, append "-" to output.

4. Append input_integer's numeric value represented in base 10 using only decimal digits to output.

5. Return output.

4.1.5. Serializing a Decimal

Given a decimal number as input_decimal, return an ASCII string suitable for use in an HTTP field value.

1. If input_decimal is not a decimal number, fail serialization.

2. If input_decimal has more than three significant digits to the right of the decimal point, round it to three decimal places, rounding the final digit to the nearest value, or to the even value if it is equidistant.

3. If input_decimal has more than 12 significant digits to the left of the decimal point after rounding, fail serialization.

4. Let output be an empty string.

5. If input_decimal is less than (but not equal to) 0, append "-" to output.

6. Append input_decimal's integer component represented in base 10 (using only decimal digits) to output; if it is zero, append "0".

7. Append "." to output.

8. If input_decimal's fractional component is zero, append "0" to output.

9. Otherwise, append the significant digits of input_decimal's fractional component represented in base 10 (using only decimal digits) to output.

10. Return output.

4.1.6. Serializing a String

Given a String as input_string, return an ASCII string suitable for use in an HTTP field value.

1. Convert input_string into a sequence of ASCII characters; if conversion fails, fail serialization.

2. If input_string contains characters in the range %x00-1f or %x7f-ff (i.e., not in VCHAR or SP), fail serialization.

3. Let output be the string DQUOTE.

4. For each character char in input_string:

    1. If char is "\" or DQUOTE:

        1. Append "\" to output.

    2. Append char to output.

5. Append DQUOTE to output.

6. Return output.

4.1.7. Serializing a Token

Given a Token as input_token, return an ASCII string suitable for use in an HTTP field value.

1. Convert input_token into a sequence of ASCII characters; if conversion fails, fail serialization.

2. If the first character of input_token is not ALPHA or "*", or the remaining portion contains a character not in tchar, ":", or "/", fail serialization.

3. Let output be an empty string.

4. Append input_token to output.

5. Return output.

4.1.8. Serializing a Byte Sequence

Given a Byte Sequence as input_bytes, return an ASCII string suitable for use in an HTTP field value.

1. If input_bytes is not a sequence of bytes, fail serialization.
2. Let output be an empty string.
3. Append ":" to output.
4. Append the result of base64-encoding input_bytes as per [RFC4648], Section 4, taking account of the requirements below.
5. Append ":" to output.
6. Return output.

The encoded data is required to be padded with "=", as per [RFC4648], Section 3.2.

Likewise, encoded data SHOULD have pad bits set to zero, as per [RFC4648], Section 3.5, unless it is not possible to do so due to implementation constraints.

4.1.9. Serializing a Boolean

Given a Boolean as input_boolean, return an ASCII string suitable for use in an HTTP field value.

1. If input_boolean is not a boolean, fail serialization.
2. Let output be an empty string.
3. Append "?" to output.
4. If input_boolean is true, append "1" to output.
5. If input_boolean is false, append "0" to output.
6. Return output.

4.1.10. Serializing a Date

Given a Date as input_date, return an ASCII string suitable for use in an HTTP field value.

1. Let output be "@".

2. Append to output the result of running Serializing an Integer with input_date (Section 4.1.4).

3. Return output.

4.1.11. Serializing a Display String

Given a sequence of Unicode codepoints as input_sequence, return an ASCII string suitable for use in an HTTP field value.

1. If input_sequence is not a sequence of Unicode codepoints, fail serialization.

2. Let byte_array be the result of applying UTF-8 encoding (Section 3 of [UTF8]) to input_sequence. If encoding fails, fail serialization.

3. Let encoded_string be a string containing "%" followed by DQUOTE.

4. For each byte in byte_array:

    1. If byte is %x25 ("%"), %x22 (DQUOTE), or in the ranges %x00-1f or %x7f-ff:

        1. Append "%" to encoded_string.

        2. Let encoded_byte be the result of applying base16 encoding (Section 8 of [RFC4648]) to byte, with any alphabetic characters converted to lowercase.

        3. Append encoded_byte to encoded_string.

    2. Otherwise, decode byte as an ASCII character and append the result to encoded_string.

5. Append DQUOTE to encoded_string.

6. Return encoded_string.

Note that [UTF8] prohibits the encoding of codepoints between U+D800 and U+DFFF (surrogates); if they occur in input_sequence, serialization will fail.

4.2. Parsing Structured Fields

When a receiving implementation parses HTTP fields that are known to be Structured Fields, it is important that care be taken, as there are a number of edge cases that can cause interoperability or even security problems. This section specifies the algorithm for doing so.

Given an array of bytes as input_bytes that represent the chosen field's field-value (which is empty if that field is not present) and field_type (one of "dictionary", "list", or "item"), return the parsed field value.

1. Convert input_bytes into an ASCII string input_string; if conversion fails, fail parsing.

2. Discard any leading SP characters from input_string.

3. If field_type is "list", let output be the result of running Parsing a List (Section 4.2.1) with input_string.

4. If field_type is "dictionary", let output be the result of running Parsing a Dictionary (Section 4.2.2) with input_string.

5. If field_type is "item", let output be the result of running Parsing an Item (Section 4.2.3) with input_string.

6. Discard any leading SP characters from input_string.

7. If input_string is not empty, fail parsing.

8. Otherwise, return output.

When generating input_bytes, parsers MUST combine all field lines in the same section (header or trailer) that case-insensitively match the field name into one comma-separated field-value, as per Section 5.2 of [HTTP]; this assures that the entire field value is processed correctly.

For Lists and Dictionaries, this has the effect of correctly concatenating all of the field's lines, as long as individual members of the top-level data structure are not split across multiple field instances. The parsing algorithms for both types allow tab characters, since these might be used to combine field lines by some implementations.

Strings split across multiple field lines will have unpredictable results, because one or more commas (with optional whitespace) will become part of the string output by the parser. Since concatenation might be done by an upstream intermediary, the results are not under the control of the serializer or the parser, even when they are both under the control of the same party.

Tokens, Integers, Decimals, and Byte Sequences cannot be split across multiple field lines because the inserted commas will cause parsing to fail.

Parsers MAY fail when processing a field value spread across multiple field lines, when one of those lines does not parse as that field. For example, a parsing handling an Example-String field that's defined as an sf-string is allowed to fail when processing this field section:

```
Example-String: "foo
Example-String: bar"
```

If parsing fails, either the entire field value MUST be ignored (i.e., treated as if the field were not present in the section), or alternatively the complete HTTP message MUST be treated as malformed. This is intentionally strict to improve interoperability and safety, and field specifications that use Structured Fields are not allowed to loosen this requirement.

Note that this requirement does not apply to an implementation that is not parsing the field; for example, an intermediary is not required to strip a failing field from a message before forwarding it.

4.2.1. Parsing a List

Given an ASCII string as input_string, return an array of (item_or_inner_list, parameters) tuples. input_string is modified to remove the parsed value.

1. Let members be an empty array.
2. While input_string is not empty:
    1. Append the result of running Parsing an Item or Inner List (Section 4.2.1.1) with input_string to members.
    2. Discard any leading OWS characters from input_string.
    3. If input_string is empty, return members.

4. Consume the first character of input_string; if it is not ",", fail parsing.

5. Discard any leading OWS characters from input_string.

6. If input_string is empty, there is a trailing comma; fail parsing.

3. No structured data has been found; return members (which is empty).

4.2.1.1. Parsing an Item or Inner List

Given an ASCII string as input_string, return the tuple (item_or_inner_list, parameters), where item_or_inner_list can be either a single bare item or an array of (bare_item, parameters) tuples. input_string is modified to remove the parsed value.

1. If the first character of input_string is "(", return the result of running Parsing an Inner List (Section 4.2.1.2) with input_string.

2. Return the result of running Parsing an Item (Section 4.2.3) with input_string.

4.2.1.2. Parsing an Inner List

Given an ASCII string as input_string, return the tuple (inner_list, parameters), where inner_list is an array of (bare_item, parameters) tuples. input_string is modified to remove the parsed value.

1. Consume the first character of input_string; if it is not "(", fail parsing.

2. Let inner_list be an empty array.

3. While input_string is not empty:

    1. Discard any leading SP characters from input_string.

    2. If the first character of input_string is ")":

        1. Consume the first character of input_string.

        2. Let parameters be the result of running Parsing Parameters (Section 4.2.3.2) with input_string.

        3. Return the tuple (inner_list, parameters).

3. Let item be the result of running Parsing an Item (Section 4.2.3) with input_string.

4. Append item to inner_list.

5. If the first character of input_string is not SP or ")", fail parsing.

4. The end of the Inner List was not found; fail parsing.

4.2.2. Parsing a Dictionary

Given an ASCII string as input_string, return an ordered map whose values are (item_or_inner_list, parameters) tuples. input_string is modified to remove the parsed value.

1. Let dictionary be an empty, ordered map.

2. While input_string is not empty:

    1. Let this_key be the result of running Parsing a Key (Section 4.2.3.3) with input_string.

    2. If the first character of input_string is "=":

        1. Consume the first character of input_string.

        2. Let member be the result of running Parsing an Item or Inner List (Section 4.2.1.1) with input_string.

    3. Otherwise:

        1. Let value be Boolean true.

        2. Let parameters be the result of running Parsing Parameters (Section 4.2.3.2) with input_string.

        3. Let member be the tuple (value, parameters).

    4. If dictionary already contains a key this_key (comparing character for character), overwrite its value with member.

    5. Otherwise, append key this_key with value member to dictionary.

    6. Discard any leading OWS characters from input_string.

    7. If input_string is empty, return dictionary.

8. Consume the first character of input_string; if it is not ",", fail parsing.

9. Discard any leading OWS characters from input_string.

10. If input_string is empty, there is a trailing comma; fail parsing.

3. No structured data has been found; return dictionary (which is empty).

Note that when duplicate Dictionary keys are encountered, all but the last instance are ignored.

4.2.3. Parsing an Item

Given an ASCII string as input_string, return a (bare_item, parameters) tuple. input_string is modified to remove the parsed value.

1. Let bare_item be the result of running Parsing a Bare Item (Section 4.2.3.1) with input_string.

2. Let parameters be the result of running Parsing Parameters (Section 4.2.3.2) with input_string.

3. Return the tuple (bare_item, parameters).

4.2.3.1. Parsing a Bare Item

Given an ASCII string as input_string, return a bare Item. input_string is modified to remove the parsed value.

1. If the first character of input_string is a "-" or a DIGIT, return the result of running Parsing an Integer or Decimal (Section 4.2.4) with input_string.

2. If the first character of input_string is a DQUOTE, return the result of running Parsing a String (Section 4.2.5) with input_string.

3. If the first character of input_string is an ALPHA or "*", return the result of running Parsing a Token (Section 4.2.6) with input_string.

4. If the first character of input_string is ":", return the result of running Parsing a Byte Sequence (Section 4.2.7) with input_string.

5. If the first character of input_string is "?", return the result of running Parsing a Boolean (Section 4.2.8) with input_string.

6. If the first character of input_string is "@", return the result of running Parsing a Date (Section 4.2.9) with input_string.

7. If the first character of input_string is "%", return the result of running Parsing a Display String (Section 4.2.10) with input_string.

8. Otherwise, the item type is unrecognized; fail parsing.

4.2.3.2. Parsing Parameters

Given an ASCII string as input_string, return an ordered map whose values are bare Items. input_string is modified to remove the parsed value.

1. Let parameters be an empty, ordered map.

2. While input_string is not empty:

    1. If the first character of input_string is not ";", exit the loop.

    2. Consume the ";" character from the beginning of input_string.

    3. Discard any leading SP characters from input_string.

    4. Let param_key be the result of running Parsing a Key (Section 4.2.3.3) with input_string.

    5. Let param_value be Boolean true.

    6. If the first character of input_string is "=":

        1. Consume the "=" character at the beginning of input_string.

        2. Let param_value be the result of running Parsing a Bare Item (Section 4.2.3.1) with input_string.

    7. If parameters already contains a key param_key (comparing character for character), overwrite its value with param_value.

    8. Otherwise, append key param_key with value param_value to parameters.

3. Return parameters.

Note that when duplicate parameter keys are encountered, all but the last instance are ignored.

4.2.3.3. Parsing a Key

Given an ASCII string as input_string, return a key. input_string is modified to remove the parsed value.

1. If the first character of input_string is not lcalpha or "*", fail parsing.

2. Let output_string be an empty string.

3. While input_string is not empty:

    1. If the first character of input_string is not one of lcalpha, DIGIT, "_", "-", ".", or "*", return output_string.

    2. Let char be the result of consuming the first character of input_string.

    3. Append char to output_string.

4. Return output_string.

4.2.4. Parsing an Integer or Decimal

Given an ASCII string as input_string, return an Integer or Decimal. input_string is modified to remove the parsed value.

NOTE: This algorithm parses both Integers (Section 3.3.1) and Decimals (Section 3.3.2), and returns the corresponding structure.

1. Let type be "integer".

2. Let sign be 1.

3. Let input_number be an empty string.

4. If the first character of input_string is "-", consume it and set sign to -1.

5. If input_string is empty, there is an empty integer; fail parsing.

6. If the first character of input_string is not a DIGIT, fail parsing.

7. While input_string is not empty:

    1. Let char be the result of consuming the first character of input_string.

    2. If char is a DIGIT, append it to input_number.

    3. Else, if type is "integer" and char is ".":

        1. If input_number contains more than 12 characters, fail parsing.

        2. Otherwise, append char to input_number and set type to "decimal".

    4. Otherwise, prepend char to input_string, and exit the loop.

    5. If type is "integer" and input_number contains more than 15 characters, fail parsing.

    6. If type is "decimal" and input_number contains more than 16 characters, fail parsing.

8. If type is "integer":

    1. Let output_number be an Integer that is the result of parsing input_number as an integer.

9. Otherwise:

    1. If the final character of input_number is ".", fail parsing.

    2. If the number of characters after "." in input_number is greater than three, fail parsing.

    3. Let output_number be a Decimal that is the result of parsing input_number as a decimal number.

10. Let output_number be the product of output_number and sign.

11. Return output_number.

4.2.5. Parsing a String

Given an ASCII string as input_string, return an unquoted String. input_string is modified to remove the parsed value.

1. Let output_string be an empty string.
2. If the first character of input_string is not DQUOTE, fail parsing.
3. Discard the first character of input_string.
4. While input_string is not empty:
    1. Let char be the result of consuming the first character of input_string.
    2. If char is a backslash ("\"):
        1. If input_string is now empty, fail parsing.
        2. Let next_char be the result of consuming the first character of input_string.
        3. If next_char is not DQUOTE or "\", fail parsing.
        4. Append next_char to output_string.
    3. Else, if char is DQUOTE, return output_string.
    4. Else, if char is in the range %x00-1f or %x7f-ff (i.e., it is not in VCHAR or SP), fail parsing.
    5. Else, append char to output_string.
5. Reached the end of input_string without finding a closing DQUOTE; fail parsing.

4.2.6. Parsing a Token

Given an ASCII string as input_string, return a Token. input_string is modified to remove the parsed value.

1. If the first character of input_string is not ALPHA or "*", fail parsing.
2. Let output_string be an empty string.

3. While input_string is not empty:

    1. If the first character of input_string is not in tchar, ":", or "/", return output_string.

    2. Let char be the result of consuming the first character of input_string.

    3. Append char to output_string.

4. Return output_string.

4.2.7. Parsing a Byte Sequence

Given an ASCII string as input_string, return a Byte Sequence. input_string is modified to remove the parsed value.

1. If the first character of input_string is not ":", fail parsing.

2. Discard the first character of input_string.

3. If there is not a ":" character before the end of input_string, fail parsing.

4. Let b64_content be the result of consuming content of input_string up to but not including the first instance of the character ":".

5. Consume the ":" character at the beginning of input_string.

6. If b64_content contains a character not included in ALPHA, DIGIT, "+", "/", and "=", fail parsing.

7. Let binary_content be the result of base64-decoding [RFC4648] b64_content, synthesizing padding if necessary (note the requirements about recipient behavior below). If base64 decoding fails, parsing fails.

8. Return binary_content.

Because some implementations of base64 do not allow rejection of encoded data that is not properly "=" padded (see [RFC4648], Section 3.2), parsers SHOULD NOT fail when "=" padding is not present, unless they cannot be configured to do so.

Because some implementations of base64 do not allow rejection of encoded data that has non-zero pad bits (see [RFC4648], Section 3.5), parsers SHOULD NOT fail when non-zero pad bits are present, unless they cannot be configured to do so.

This specification does not relax the requirements in Sections 3.1 and 3.3 of [RFC4648]; therefore, parsers MUST fail on characters outside the base64 alphabet and on line feeds in encoded data.

4.2.8. Parsing a Boolean

Given an ASCII string as input_string, return a Boolean. input_string is modified to remove the parsed value.

1. If the first character of input_string is not "?", fail parsing.
2. Discard the first character of input_string.
3. If the first character of input_string matches "1", discard the first character, and return true.
4. If the first character of input_string matches "0", discard the first character, and return false.
5. No value has matched; fail parsing.

4.2.9. Parsing a Date

Given an ASCII string as input_string, return a Date. input_string is modified to remove the parsed value.

1. If the first character of input_string is not "@", fail parsing.
2. Discard the first character of input_string.
3. Let output_date be the result of running Parsing an Integer or Decimal (Section 4.2.4) with input_string.
4. If output_date is a Decimal, fail parsing.
5. Return output_date.

4.2.10. Parsing a Display String

Given an ASCII string as input_string, return a sequence of Unicode codepoints. input_string is modified to remove the parsed value.

1. If the first two characters of input_string are not "%" followed by DQUOTE, fail parsing.

2. Discard the first two characters of input_string.

3. Let byte_array be an empty byte array.

4. While input_string is not empty:

    1. Let char be the result of consuming the first character of input_string.

    2. If char is in the range %x00-1f or %x7f-ff (i.e., it is not in VCHAR or SP), fail parsing.

    3. If char is "%":

        1. Let octet_hex be the result of consuming two characters from input_string. If there are not two characters, fail parsing.

        2. If octet_hex contains characters outside the range %x30-39 or %x61-66 (i.e., it is not in 0-9 or lowercase a-f), fail parsing.

        3. Let octet be the result of hex decoding octet_hex (Section 8 of [RFC4648]).

        4. Append octet to byte_array.

    4. If char is DQUOTE:

        1. Let unicode_sequence be the result of decoding byte_array as a UTF-8 string (Section 3 of [UTF8]). Fail parsing if decoding fails.

        2. Return unicode_sequence.

    5. Otherwise, if char is not "%" or DQUOTE:

        1. Let byte be the result of applying ASCII encoding to char.

        2. Append byte to byte_array.

5. Reached the end of input_string without finding a closing DQUOTE; fail parsing.

5. IANA Considerations

Please add the following note to the "Hypertext Transfer Protocol (HTTP) Field Name Registry":

> The "Structured Type" column indicates the type of the field (per RFC nnnn), if any, and may be "Dictionary", "List" or "Item".
>
> Note that field names beginning with characters other than ALPHA or "*" will not be able to be represented as a Structured Fields Token, and therefore may be incompatible with being mapped into field values that refer to it.

Then, add a new column, "Structured Type".

Then, add the indicated Structured Type for each existing registry entry listed in Table 1.

Field Name	Structured Type
Accept-CH	List
Cache-Status	List
CDN-Cache-Control	Dictionary
Cross-Origin-Embedder-Policy	Item
Cross-Origin-Embedder-Policy-Report-Only	Item
Cross-Origin-Opener-Policy	Item
Cross-Origin-Opener-Policy-Report-Only	Item
Origin-Agent-Cluster	Item
Priority	Dictionary
Proxy-Status	List

Table 1: Existing Fields

6. Security Considerations

The size of most types defined by Structured Fields is not limited; as a result, extremely large fields could be an attack vector (e.g., for resource consumption). Most HTTP implementations limit the sizes of individual fields as well as the overall header or trailer section size to mitigate such attacks.

It is possible for parties with the ability to inject new HTTP fields to change the meaning of a Structured Field. In some circumstances, this will cause parsing to fail, but it is not possible to reliably fail in all such circumstances.

The Display String type can convey any possible Unicode code point without sanitization; for example, they might contain unassigned code points, control points (including NUL), or noncharacters. Therefore, applications consuming Display Strings need to consider strategies such as filtering or escaping untrusted content before displaying it. See [PRECIS] and [UNICODE-SECURITY].

7. References

7.1. Normative References

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

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/rfc/rfc20>.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

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

[UTF8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003, <http://www.rfc-editor.org/info/std63>.

7.2. Informative References

[HPACK] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/rfc/rfc7541>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2019, DOI 10.1109/IEEESTD.2019.8766229, ISBN 978-1-5044-5924-2, July 2019, <https://ieeexplore.ieee.org/document/8766229>.

[PRECIS] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 8264, DOI 10.17487/RFC8264, October 2017, <https://www.rfc-editor.org/rfc/rfc8264>.

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

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/rfc/rfc7493>.

[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/rfc/rfc8259>.

[UNICODE-SECURITY] Davis, M. and M. Suignard, "Unicode Security Considerations", Unicode Technical Report #16, 19 September 2014, <http://www.unicode.org/reports/tr36/>.

Appendix A. Frequently Asked Questions

A.1. Why Not JSON?

Earlier proposals for Structured Fields were based upon JSON [RFC8259]. However, constraining its use to make it suitable for HTTP fields required senders and recipients to implement specific additional handling.

For example, JSON has specification issues around large numbers and objects with duplicate members. Although advice for avoiding these issues is available (e.g., [RFC7493]), it cannot be relied upon.

Likewise, JSON strings are by default Unicode strings, which have a number of potential interoperability issues (e.g., in comparison). Although implementers can be advised to avoid non-ASCII content where unnecessary, this is difficult to enforce.

Another example is JSON's ability to nest content to arbitrary depths. Since the resulting memory commitment might be unsuitable (e.g., in embedded and other limited server deployments), it's necessary to limit it in some fashion; however, existing JSON implementations have no such limits, and even if a limit is specified, it's likely that some field definition will find a need to violate it.

Because of JSON's broad adoption and implementation, it is difficult to impose such additional constraints across all implementations; some deployments would fail to enforce them, thereby harming interoperability. In short, if it looks like JSON, people will be tempted to use a JSON parser/serializer on field values.

Since a major goal for Structured Fields is to improve interoperability and simplify implementation, these concerns led to a format that requires a dedicated parser and serializer.

Additionally, there were widely shared feelings that JSON doesn't "look right" in HTTP fields.

Appendix B. Implementation Notes

A generic implementation of this specification should expose the top-level serialize (Section 4.1) and parse (Section 4.2) functions. They need not be functions; for example, it could be implemented as an object, with methods for each of the different top-level types.

For interoperability, it's important that generic implementations be complete and follow the algorithms closely; see Section 1.1. To aid this, a common test suite is being maintained by the community at <https://github.com/httpwg/structured-field-tests>.

Implementers should note that Dictionaries and Parameters are order-preserving maps. Some fields may not convey meaning in the ordering of these data types, but it should still be exposed so that it will be available to applications that need to use it.

Likewise, implementations should note that it's important to preserve the distinction between Tokens and Strings. While most programming languages have native types that map to the other types well, it may be necessary to create a wrapper "token" object or use a parameter on functions to assure that these types remain separate.

The serialization algorithm is defined in a way that it is not strictly limited to the data types defined in Section 3 in every case. For example, Decimals are designed to take broader input and round to allowed values.

Implementations are allowed to limit the size of different structures, subject to the minimums defined for each type. When a structure exceeds an implementation limit, that structure fails parsing or serialization.

Appendix C. ABNF

This section uses the Augmented Backus-Naur Form (ABNF) notation [RFC5234] to illustrate expected syntax of Structured Fields. However, it cannot be used to validate their syntax, because it does not capture all requirements.

This section is non-normative. If there is disagreement between the parsing algorithms and ABNF, the specified algorithms take precedence.

```
sf-list       = list-member *( OWS "," OWS list-member )
list-member   = sf-item / inner-list

inner-list    = "(" *SP [ sf-item *( 1*SP sf-item ) *SP ] ")"
                parameters

parameters    = *( ";" *SP parameter )
parameter     = param-key [ "=" param-value ]
param-key     = key
key           = ( lcalpha / "*" )
                *( lcalpha / DIGIT / "_" / "-" / "." / "*" )
lcalpha       = %x61-7A ; a-z
param-value   = bare-item

sf-dictionary = dict-member *( OWS "," OWS dict-member )
dict-member   = member-key ( parameters / ( "=" member-value ))
member-key    = key
member-value  = sf-item / inner-list

sf-item   = bare-item parameters
bare-item = sf-integer / sf-decimal / sf-string / sf-token
            / sf-binary / sf-boolean / sf-date / sf-displaystring

sf-integer       = ["-"] 1*15DIGIT
sf-decimal       = ["-"] 1*12DIGIT "." 1*3DIGIT
sf-string        = DQUOTE *( unescaped / "%" / bs-escaped ) DQUOTE
sf-token         = ( ALPHA / "*" ) *( tchar / ":" / "/" )
sf-binary        = ":" base64 ":"
sf-boolean       = "?" ( "0" / "1" )
sf-date          = "@" sf-integer
sf-displaystring = "%" DQUOTE *( unescaped / "\" / pct-encoded )
                   DQUOTE

base64       = *( ALPHA / DIGIT / "+" / "/" ) *"="

unescaped    = %x20-21 / %x23-24 / %x26-5B / %x5D-7E
bs-escaped   = "\" ( DQUOTE / "\" )

pct-encoded  = "%" lc-hexdig lc-hexdig
lc-hexdig = DIGIT / %x61-66 ; 0-9, a-f
```

Appendix D. Changes from RFC 8941

This revision of the Structured Field Values for HTTP specification has made the following changes:

* Added the Date structured type. (Section 3.3.7)

* Stopped encouraging use of ABNF in definitions of new structured fields. (Section 2)

* Moved ABNF to an informative appendix. (Appendix C)

* Added a "Structured Type" column to the HTTP Field Name Registry. (Section 5)

* Refined parse failure handling. (Section 4.2)

* Added the Display String structured type. (Section 3.3.8)

Acknowledgements

Many thanks to Matthew Kerwin for his detailed feedback and careful consideration during the development of this specification.

Thanks also to Ian Clelland, Roy Fielding, Anne van Kesteren, Kazuho Oku, Evert Pot, Julian Reschke, Martin Thomson, Mike West, and Jeffrey Yasskin for their contributions.

Authors' Addresses

Mark Nottingham
Cloudflare
Prahran VIC
Australia
Email: mnot@mnot.net
URI: https://www.mnot.net/

Poul-Henning Kamp
The Varnish Cache Project
Email: phk@varnish-cache.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 21 October 2024

R. Housley
Vigil Security

19 April 2024

Use of the SHA3 One-way Hash Functions in the Cryptographic Message Syntax (CMS)
draft-ietf-lamps-cms-sha3-hash-03

Abstract

This document describes the conventions for using the one-way hash functions in the SHA3 family with the Cryptographic Message Syntax (CMS). The SHA3 family can be used as a message digest algorithm, as part of a signature algorithm, as part of a message authentication code, or part of a key derivation function.

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 21 October 2024.

Table of Contents

1. Introduction

The Cryptographic Message Syntax (CMS) [RFC5652] is used to digitally sign, digest, authenticate, or encrypt arbitrary message contents. This specification describes the use of the four one-way hash functions in the SHA3 family (SHA3-224, SHA3-256, SHA3-384, and SHA3-512) [SHA3] with the CMS. In addition, this specification describes the use of these four one-way hash functions with the RSASSA PKCS#1 version 1.5 signature algorithm [RFC8017] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [DSS] with the CMS signed-data content type.

This document should not be confused with RFC 8702 [RFC8702], which defines conventions for using the the SHAKE family of SHA3-based extensible output functions with the CMS.

1.1. ASN.1

CMS values are generated using ASN.1 [X.680], using the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X.690].

1.2. Terminology

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.

2. Message Digest Algorithms

One-way hash functions are also referred to as message digest algorithms.
This section specifies the conventions employed by CMS implementations that support SHA3-224, SHA3-256, SHA3-384, and SHA3-512 [SHA3].

Digest algorithm identifiers are located in the SignedData digestAlgorithms field, the SignerInfo digestAlgorithm field, the DigestedData digestAlgorithm field, and the AuthenticatedData digestAlgorithm field.

Digest values are located in the DigestedData digest field and the Message Digest authenticated attribute. In addition, digest values are input to signature algorithms.

SHA3-224, SHA3-256, SHA3-384, and SHA3-512 produce output values with 224, 256, 384, and 512 bits, respectively. The object identifiers for these four one-way hash functions are as follows:

```
hashAlgs OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101) csor(3) nistAlgorithm(4) 2 }

id-sha3-224 OBJECT IDENTIFIER ::= { hashAlgs 7 }

id-sha3-256 OBJECT IDENTIFIER ::= { hashAlgs 8 }

id-sha3-384 OBJECT IDENTIFIER ::= { hashAlgs 9 }

id-sha3-512 OBJECT IDENTIFIER ::= { hashAlgs 10 }
```

When using the id-sha3-224, id-sha3-s256, id-sha3-384, or id-sha3-512 algorithm identifiers, the parameters field MUST be absent; not NULL but absent.

3. Signature Algorithms

This section specifies the conventions employed by CMS implementations that support the four SHA3 one-way hash functions with the RSASSA PKCS#1 version 1.5 signature algorithm [RFC8017] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [DSS] with the CMS signed-data content type.

Signature algorithm identifiers are located in the SignerInfo signatureAlgorithm field of SignedData. Also, signature algorithm identifiers are located in the SignerInfo signatureAlgorithm field of countersignature attributes.

Signature values are located in the SignerInfo signature field of SignedData. Also, signature values are located in the SignerInfo signature field of countersignature attributes.

3.1. RSASSA PKCS#1 v1.5 with SHA3

The RSASSA PKCS#1 v1.5 is defined in [RFC8017]. When RSASSA PKCS#1 v1.5 is used in conjunction with one of the SHA3 one-way hash functions, the object identifiers are:

```
OID ::= OBJECT IDENTIFIER

sigAlgs OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101) csor(3) nistAlgorithm(4) 3 }

id-rsassa-pkcs1-v1-5-with-sha3-224 OID ::= { sigAlgs 13 }

id-rsassa-pkcs1-v1-5-with-sha3-256 OID ::= { sigAlgs 14 }

id-rsassa-pkcs1-v1-5-with-sha3-384 OID ::= { sigAlgs 15 }

id-rsassa-pkcs1-v1-5-with-sha3-512 OID ::= { sigAlgs 16 }
```

The algorithm identifier for RSASSA PKCS#1 v1.5 subject public keys in certificates is specified in [RFC3279], and it is repeated here for convenience:

```
rsaEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 }
```

When the rsaEncryption, id-rsassa-pkcs1-v1-5-with-sha3-224, id-rsassa-pkcs1-v1-5-with-sha3-256, id-rsassa-pkcs1-v1-5-with-sha3-384, and id-rsassa-pkcs1-v1-5-with-sha3-512 algorithm identifier is used, AlgorithmIdentifier parameters field MUST contain NULL.

When the rsaEncryption algorithm identifier is used, the RSA public key, which is composed of a modulus and a public exponent, MUST be encoded using the RSAPublicKey type as specified in [RFC3279]. The output of this encoding is carried in the certificate subject public key. The definition of RSAPublicKey is repeated here for convenience:

```
RSAPublicKey ::= SEQUENCE {
   modulus INTEGER, -- n
   publicExponent INTEGER } -- e
```

When signing, the RSASSA PKCS#1 v1.5 signature algorithm generates a single value, and that value is used directly as the signature value.

3.2. ECDSA with SHA3

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in [DSS]. When ECDSA is used in conjunction with one of the SHA3 one-way hash functions, the object identifiers are:

```
sigAlgs OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101) csor(3) nistAlgorithm(4) 3 }

id-ecdsa-with-sha3-224 OBJECT IDENTIFIER ::= { sigAlgs 9 }

id-ecdsa-with-sha3-256 OBJECT IDENTIFIER ::= { sigAlgs 10 }

id-ecdsa-with-sha3-384 OBJECT IDENTIFIER ::= { sigAlgs 11 }

id-ecdsa-with-sha3-512 OBJECT IDENTIFIER ::= { sigAlgs 12 }
```

When using the id-ecdsa-with-sha3-224, id-ecdsa-with-sha3-256, id-ecdsa-with-sha3-384, and id-ecdsa-with-sha3-512 algorithm identifiers, the parameters field MUST be absent; not NULL but absent.

The conventions for ECDSA public keys is as specified in [RFC5480]. The ECParameters associated with the ECDSA public key in the signers certificate SHALL apply to the verification of the signature.

When signing, the ECDSA algorithm generates two values. These values are commonly referred to as r and s. To easily transfer these two values as one signature, they MUST be ASN.1 encoded using the ECDSA-Sig-Value defined in [RFC3279] and repeated here for convenience:

```
ECDSA-Sig-Value ::= SEQUENCE {
    r  INTEGER,
    s  INTEGER }
```

4. Message Authentication Codes using HMAC and SHA3

This section specifies the conventions employed by CMS implementations that support the HMAC [RFC2104] with SHA3 message authentication code (MAC).

MAC algorithm identifiers are located in the AuthenticatedData macAlgorithm field.

MAC values are located in the AuthenticatedData mac field.

When HMAC is used in conjunction with one of the SHA3 one-way hash functions, the object identifiers are:

```
hashAlgs OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101) csor(3) nistAlgorithm(4) 2 }

id-hmacWithSHA3-224 OBJECT IDENTIFIER ::= { hashAlgs 13 }

id-hmacWithSHA3-256 OBJECT IDENTIFIER ::= { hashAlgs 14 }

id-hmacWithSHA3-384 OBJECT IDENTIFIER ::= { hashAlgs 15 }

id-hmacWithSHA3-512 OBJECT IDENTIFIER ::= { hashAlgs 16 }
```

When the id-hmacWithSHA3-224, id-hmacWithSHA3-256, id-hmacWithSHA3-384, and id-hmacWithSHA3-512 algorithm identifier is used, the parameters field MUST be absent; not NULL but absent.

5. Key Derivation Functions

The CMS KEMRecipientInfo structure [I-D.ietf-lamps-cms-kemri] is one place where algorithm identifiers for key-derivation functions are needed.

5.1. HKDF with SHA3

This section assigns four algorithm identifiers that can be employed by CMS implementations that support the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [RFC5869] with the SHA3 family of hash functions.

When HKDF is used in conjunction with one of the SHA3 one-way hash functions, the object identifiers are:

```
id-alg OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) 3 }

id-alg-hkdf-with-sha3-224 OBJECT IDENTIFIER ::= { id-alg TBD1 }

id-alg-hkdf-with-sha3-256 OBJECT IDENTIFIER ::= { id-alg TBD2 }

id-alg-hkdf-with-sha3-384 OBJECT IDENTIFIER ::= { id-alg TBD3 }

id-alg-hkdf-with-sha3-512 OBJECT IDENTIFIER ::= { id-alg TBD4 }
```

When id-alg-hkdf-with-sha3-224, id-alg-hkdf-with-sha3-256, id-alg-hkdf-with-sha3-384, or id-alg-hkdf-with-sha3-512 is used in an algorithm identifier, the parameters field MUST be absent; not NULL but absent.

5.2. KMAC128-KDF and KMAC256-KDF

This section specifies the conventions employed by CMS implementations that employ either the KMAC128 or KMAC256 as a key derivation function as defined in Section 4.4 of [NIST.SP.800-108r1-upd1].

KMAC128 and KMAC256 are specified in [NIST.SP.800-185]. The use of KMAC128 and KMAC256 as a key derivation function are defined as:

KMAC128-KDF is KMAC128(K, X, L, S).

KMAC256-KDF is KMAC256(K, X, L, S).

The parameters are:

K the input key-derivation key. The length of K MUST be less than 2^2040.

X the context, which contains the ASN.1 DER encoding of CMSORIforKEMOtherInfo when the KDF is used with [I-D.ietf-lamps-cms-kemri].

L the output length, in bits. L MUST be greater than or equal to 0, and L MUST be less than 2^2040.

S the optional customization label, such as "KDF" (0x4B4446). The length of S MUST be less than 2^2040.

When KMAC128-KDF or KMAC256-KDF is used, the object identifiers are:

```
hashAlgs OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101) csor(3) nistAlgorithm(4) 2 }

id-kmac128 OBJECT IDENTIFIER ::= { hashAlgs 21 }

id-kmac256 OBJECT IDENTIFIER ::= { hashAlgs 22 }
```

When the id-kmac128 or id-kmac256 is used as part of an algorithm identifier, the parameters field MUST be absent if no customization label is used for S. If any other value is used for S, then parameters field MUST be present and contain the value of S, encoded as Customization.

```
Customization ::= OCTET STRING
```

5.3. KDF2 and KDF3 with SHA3

This section specifies the conventions employed by CMS implementations that employ either the KDF2 or KDF3 functions defined in [ANS-X9.44]. The CMS KEMRecipientInfo structure [I-D.ietf-lamps-cms-kemri] is one place where algorithm identifiers for key-derivation functions are needed.

When KDF2 and KDF3 are used, they are identified by the id-kdf-kdf2 and id-kdf-kdf3 object identifiers, respectively. The algorithm identifier parameters carries an algorithm identifier to indicate which hash function is being employed. To support SHA3, an algorithm identifier from Section 2 is carried in the parameter.

```
x9-44 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
    tc68(133) country(16) x9(840) x9Standards(9) x9-44(44) }

x9-44-components OBJECT IDENTIFIER ::= { x9-44 components(1) }

id-kdf-kdf2 OBJECT IDENTIFIER ::= { x9-44-components kdf2(1) }

id-kdf-kdf3 OBJECT IDENTIFIER ::= { x9-44-components kdf3(2) }
```

6. Security Considerations

Implementations must protect the signer’s private key. Compromise of the signer’s private key permits masquerade.

When more than two parties share the same message-authentication key, data origin authentication is not provided. Any party that knows the message-authentication key can compute a valid MAC, therefore the content could originate from any one of the parties.

Implementations must randomly generate message-authentication keys and one-time values, such as the k value when generating a ECDSA signature. In addition, the generation of public/private key pairs relies on a random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate cryptographic values can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult. RFC 4086 [RFC4086] offers important guidance in this area, and Appendix 3 of FIPS Pub 186-4 [DSS] provides some PRNG techniques.

Implementers should be aware that cryptographic algorithms become weaker with time. As new cryptanalysis techniques are developed and computing performance improves, the work factor to break a particular cryptographic algorithm will reduce. Therefore, cryptographic algorithm implementations should be modular allowing new algorithms to be readily inserted. That is, implementers should be prepared to regularly update the set of algorithms in their implementations.

7. IANA Considerations

IANA is asked to assign one object identifier for the ASN.1 module in Appendix "Appendix. ASN.1 Module" in the "SMI Security for S/MIME Module Identifiers (1.2.840.113549.1.9.16.0)" registry [IANA-MOD]:

```
id-mod-sha3-oids-2023 OBJECT IDENTIFIER ::= {
   iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
   pkcs-9(9) smime(16) mod(0) TBD0 }
```

IANA is asked to assign four object identifiers for the HKDF using SHA3 algorithm identifiers in the "SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3)" registry [IANA-ALG]:

```
id-alg-hkdf-with-sha3-224 OBJECT IDENTIFIER ::= { id-alg TBD1 }

id-alg-hkdf-with-sha3-256 OBJECT IDENTIFIER ::= { id-alg TBD2 }

id-alg-hkdf-with-sha3-384 OBJECT IDENTIFIER ::= { id-alg TBD3 }

id-alg-hkdf-with-sha3-512 OBJECT IDENTIFIER ::= { id-alg TBD4 }
```

Acknowledgements

Thanks to Daniel Van Geest and Sean Turner for their careful review and thoughtful comments.

Thanks to Sara Kerman, Quynh Dang, and David Cooper for getting the object identifiers assigned for KMAC128 and KMAC256.

References

Normative References

[ANS-X9.44]
American National Standards Institute, "Public Key Cryptography for the Financial Services Industry -- Key Establishment Using Integer Factorization Cryptography", American National Standard X9.44, 2007.

[DSS] National Institute of Standards and Technology, "Digital Signature Standard (DSS) version 4", FIPS PUB 186-4, July 2013, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[NIST.SP.800-108r1-upd1]
National Institute of Standards and Technology, "Recommendation for key derivation using pseudorandom functions", DOI 10.6028/NIST.SP.800-108r1-upd1, 2 February 2024, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-108r1-upd1.pdf>.

[NIST.SP.800-185]
National Institute of Standards and Technology, "SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash", DOI 10.6028/NIST.SP.800-185, December 2016, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

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

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, DOI 10.17487/RFC3279, April 2002, <https://www.rfc-editor.org/rfc/rfc3279>.

[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/rfc/rfc5480>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/rfc/rfc5652>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/rfc/rfc5912>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/rfc/rfc8017>.

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

[SHA3] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS PUB 202, August 2015, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.

[X.680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

[X.690] ITU-T, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

Informative References

[I-D.ietf-lamps-cms-kemri]
Housley, R., Gray, J., and T. Okubo, "Using Key Encapsulation Mechanism (KEM) Algorithms in the Cryptographic Message Syntax (CMS)", Work in Progress, Internet-Draft, draft-ietf-lamps-cms-kemri-08, 6 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-cms-kemri-08>.

[IANA-ALG] IANA, "SMI Security for for S/MIME Algorithms (1.2.840.113549.1.9.16.3)", n.d., <https://www.iana.org/assignments/smi-numbers/>.

[IANA-MOD] IANA, "SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0)", n.d., <https://www.iana.org/assignments/smi-numbers/>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC8702] Kampanakis, P. and Q. Dang, "Use of the SHAKE One-Way Hash Functions in the Cryptographic Message Syntax (CMS)", RFC 8702, DOI 10.17487/RFC8702, January 2020, <https://www.rfc-editor.org/rfc/rfc8702>.

Appendix. ASN.1 Module

This section contains the ASN.1 module for the algorithm identifiers using SHA3 family of hash functions [SHA3]. This module imports types from other ASN.1 modules that are defined in [RFC5912].

<CODE BEGINS>

```
SHA3-OIDs-2023
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    smime(16) modules(0) id-mod-sha3-oids-2023(TBD0) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

EXPORTS ALL;

IMPORTS

  AlgorithmIdentifier{}, DIGEST-ALGORITHM, SIGNATURE-ALGORITHM,
  KEY-DERIVATION, MAC-ALGORITHM
  FROM AlgorithmInformation-2009  -- [RFC5912]
    { iso(1) identified-organization(3) dod(6) internet(1)
```

```
            security(5) mechanisms(5) pkix(7) id-mod(0)
            id-mod-algorithmInformation-02(58) }

       mda-sha1, pk-rsa, pk-ec, ECDSA-Sig-Value
       FROM PKIXAlgs-2009  -- [RFC5912]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) id-mod(0)
           id-mod-pkix1-algorithms2008-02(56) }

       mda-sha224, mda-sha256, mda-sha384, mda-sha512
       FROM PKIX1-PSS-OAEP-Algorithms-2009  -- [RFC5912]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) id-mod(0)
           id-mod-pkix1-rsa-pkalgs-02(54) } ;


      --
      -- Alias
      --

      OID ::= OBJECT IDENTIFIER


      --
      -- Object Identifier Arcs
      --

      nistAlgorithm OID ::= { joint-iso-itu-t(2) country(16)
          us(840) organization(1) gov(101) csor(3) 4 }

      hashAlgs OID ::= { nistAlgorithm 2 }

      sigAlgs OID ::= { nistAlgorithm 3 }

      x9-44 OID ::= { iso(1) identified-organization(3) tc68(133)
          country(16) x9(840) x9Standards(9) x9-44(44) }

      x9-44-components OID ::= { x9-44 components(1) }

      id-alg OID ::= { iso(1) member-body(2) us(840)
          rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) 3 }


      --
      -- Message Digest Algorithms
      --

      id-sha3-224 OID ::= { hashAlgs 7 }
```

```
id-sha3-256 OID ::= { hashAlgs 8 }

id-sha3-384 OID ::= { hashAlgs 9 }

id-sha3-512 OID ::= { hashAlgs 10 }

mda-sha3-224 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-sha3-224
    PARAMS ARE absent }

mda-sha3-256 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-sha3-256
    PARAMS ARE absent }

mda-sha3-384 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-sha3-384
    PARAMS ARE absent }

mda-sha3-512 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-sha3-512
    PARAMS ARE absent }

HashAlgorithm ::= AlgorithmIdentifier{ DIGEST-ALGORITHM,
                      { HashAlgorithms } }

HashAlgorithms DIGEST-ALGORITHM ::=  {
    mda-sha3-224 |
    mda-sha3-256 |
    mda-sha3-384 |
    mda-sha3-512,
    ... }


--
-- Signature Algorithms
--

id-rsassa-pkcs1-v1-5-with-sha3-224 OID ::= { sigAlgs 13 }

id-rsassa-pkcs1-v1-5-with-sha3-256 OID ::= { sigAlgs 14 }

id-rsassa-pkcs1-v1-5-with-sha3-384 OID ::= { sigAlgs 15 }

id-rsassa-pkcs1-v1-5-with-sha3-512 OID ::= { sigAlgs 16 }

id-ecdsa-with-sha3-224 OID ::= { sigAlgs 9 }

id-ecdsa-with-sha3-256 OID ::= { sigAlgs 10 }
```

```
id-ecdsa-with-sha3-384 OID ::= { sigAlgs 11 }

id-ecdsa-with-sha3-512 OID ::= { sigAlgs 12 }

sa-rsaWithSHA3-224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-rsassa-pkcs1-v1-5-with-sha3-224
    PARAMS TYPE NULL ARE required
    HASHES { mda-sha3-224 }
    PUBLIC-KEYS { pk-rsa }
    SMIME-CAPS {IDENTIFIED BY
        id-rsassa-pkcs1-v1-5-with-sha3-224 } }

sa-rsaWithSHA3-256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-rsassa-pkcs1-v1-5-with-sha3-256
    PARAMS TYPE NULL ARE required
    HASHES { mda-sha3-256 }
    PUBLIC-KEYS { pk-rsa }
    SMIME-CAPS {IDENTIFIED BY
        id-rsassa-pkcs1-v1-5-with-sha3-256 } }

sa-rsaWithSHA3-384 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-rsassa-pkcs1-v1-5-with-sha3-384
    PARAMS TYPE NULL ARE required
    HASHES { mda-sha3-384 }
    PUBLIC-KEYS { pk-rsa }
    SMIME-CAPS {IDENTIFIED BY
        id-rsassa-pkcs1-v1-5-with-sha3-384 } }

sa-rsaWithSHA3-512 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-rsassa-pkcs1-v1-5-with-sha3-512
    PARAMS TYPE NULL ARE required
    HASHES { mda-sha3-512 }
    PUBLIC-KEYS { pk-rsa }
    SMIME-CAPS {IDENTIFIED BY
        id-rsassa-pkcs1-v1-5-with-sha3-512 } }

sa-ecdsaWithSHA3-224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-ecdsa-with-sha3-224
    VALUE ECDSA-Sig-Value
    PARAMS ARE absent
    HASHES { mda-sha3-224 }
    PUBLIC-KEYS { pk-ec }
    SMIME-CAPS {IDENTIFIED BY id-ecdsa-with-sha3-224 } }

sa-ecdsaWithSHA3-256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-ecdsa-with-sha3-256
    VALUE ECDSA-Sig-Value
    PARAMS ARE absent
```

```
        HASHES { mda-sha3-256 }
        PUBLIC-KEYS { pk-ec }
        SMIME-CAPS {IDENTIFIED BY id-ecdsa-with-sha3-256 } }

    sa-ecdsaWithSHA3-384 SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-ecdsa-with-sha3-384
        VALUE ECDSA-Sig-Value
        PARAMS ARE absent
        HASHES { mda-sha3-384 }
        PUBLIC-KEYS { pk-ec }
        SMIME-CAPS {IDENTIFIED BY id-ecdsa-with-sha3-384 } }

    sa-ecdsaWithSHA3-512 SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-ecdsa-with-sha3-512
        VALUE ECDSA-Sig-Value
        PARAMS ARE absent
        HASHES { mda-sha3-512 }
        PUBLIC-KEYS { pk-ec }
        SMIME-CAPS {IDENTIFIED BY id-ecdsa-with-sha3-512 } }

    SignatureAlg ::= AlgorithmIdentifier{ SIGNATURE-ALGORITHM,
                          { SignatureAlgs } }

    SignatureAlgs SIGNATURE-ALGORITHM ::= {
        sa-rsaWithSHA3-224 |
        sa-rsaWithSHA3-256 |
        sa-rsaWithSHA3-384 |
        sa-rsaWithSHA3-512 |
        sa-ecdsaWithSHA3-224 |
        sa-ecdsaWithSHA3-256 |
        sa-ecdsaWithSHA3-384 |
        sa-ecdsaWithSHA3-512,
        ... }


    --
    -- Message Authentication Codes
    --

    id-hmacWithSHA3-224 OID ::= { hashAlgs 13 }

    id-hmacWithSHA3-256 OID ::= { hashAlgs 14 }

    id-hmacWithSHA3-384 OID ::= { hashAlgs 15 }

    id-hmacWithSHA3-512 OID ::= { hashAlgs 16 }

    maca-hmacWithSHA3-224 MAC-ALGORITHM ::= {
```

```
        IDENTIFIER id-hmacWithSHA3-224
        PARAMS ARE absent
        IS-KEYED-MAC TRUE
        SMIME-CAPS {IDENTIFIED BY id-hmacWithSHA3-224 } }

    maca-hmacWithSHA3-256 MAC-ALGORITHM ::= {
        IDENTIFIER id-hmacWithSHA3-256
        PARAMS ARE absent
        IS-KEYED-MAC TRUE
        SMIME-CAPS {IDENTIFIED BY id-hmacWithSHA3-256 } }

    maca-hmacWithSHA3-384 MAC-ALGORITHM ::= {
        IDENTIFIER id-hmacWithSHA3-384
        PARAMS ARE absent
        IS-KEYED-MAC TRUE
        SMIME-CAPS {IDENTIFIED BY id-hmacWithSHA3-384 } }

    maca-hmacWithSHA3-512 MAC-ALGORITHM ::= {
        IDENTIFIER id-hmacWithSHA3-512
        PARAMS ARE absent
        IS-KEYED-MAC TRUE
        SMIME-CAPS {IDENTIFIED BY id-hmacWithSHA3-512 } }

    MACAlgorithm ::= AlgorithmIdentifier{ MAC-ALGORITHM,
                        { MACAlgorithms } }

    MACAlgorithms MAC-ALGORITHM ::= {
        maca-hmacWithSHA3-224 |
        maca-hmacWithSHA3-256 |
        maca-hmacWithSHA3-384 |
        maca-hmacWithSHA3-512,
        ... }


    --
    -- Key Derivation Algorithms
    --

    id-alg-hkdf-with-sha3-224 OID ::= { id-alg TBD1 }

    id-alg-hkdf-with-sha3-256 OID ::= { id-alg TBD2 }

    id-alg-hkdf-with-sha3-384 OID ::= { id-alg TBD3 }

    id-alg-hkdf-with-sha3-512 OID ::= { id-alg TBD4 }

    id-kmac128 OID ::= { hashAlgs 21 }
```

```
id-kmac256  OID ::= { hashAlgs 22 }

id-kdf-kdf2 OID ::= { x9-44-components kdf2(1) }

id-kdf-kdf3 OID ::= { x9-44-components kdf3(2) }

kda-hkdf-with-sha3-224 KEY-DERIVATION ::= {
    IDENTIFIER id-alg-hkdf-with-sha3-224
    PARAMS ARE absent
    -- No S/MIME caps defined -- }

kda-hkdf-with-sha3-256 KEY-DERIVATION ::= {
    IDENTIFIER id-alg-hkdf-with-sha3-256
    PARAMS ARE absent
    -- No S/MIME caps defined -- }

kda-hkdf-with-sha3-384 KEY-DERIVATION ::= {
    IDENTIFIER id-alg-hkdf-with-sha3-384
    PARAMS ARE absent
    -- No S/MIME caps defined -- }

kda-hkdf-with-sha3-512 KEY-DERIVATION ::= {
    IDENTIFIER id-alg-hkdf-with-sha3-512
    PARAMS ARE absent
    -- No S/MIME caps defined -- }

kda-kmac128 KEY-DERIVATION ::= {
    IDENTIFIER id-kmac128
    PARAMS TYPE Customization ARE optional
    -- PARAMS are absent when Customization is ''H --
    -- No S/MIME caps defined -- }

kda-kmac256 KEY-DERIVATION ::= {
    IDENTIFIER id-kmac256
    PARAMS TYPE Customization ARE optional
    -- PARAMS are absent when Customization is ''H --
    -- No S/MIME caps defined -- }

kda-kdf2 KEY-DERIVATION ::= {
    IDENTIFIER id-kdf-kdf2
    PARAMS TYPE KDF2-HashFunction ARE required
    -- No S/MIME caps defined -- }

kda-kdf3 KEY-DERIVATION ::= {
    IDENTIFIER id-kdf-kdf3
    PARAMS TYPE KDF3-HashFunction ARE required
    -- No S/MIME caps defined -- }
```

```
      Customization ::= OCTET STRING

      KDF2-HashFunction ::= AlgorithmIdentifier { DIGEST-ALGORITHM,
                                { KDF2-HashFunctions } }

      KDF2-HashFunctions DIGEST-ALGORITHM ::= {
         X9-HashFunctions,
         ... }

      KDF3-HashFunction ::= AlgorithmIdentifier { DIGEST-ALGORITHM,
                                { KDF3-HashFunctions } }

      KDF3-HashFunctions DIGEST-ALGORITHM ::= {
         X9-HashFunctions,
         ... }

      X9-HashFunctions DIGEST-ALGORITHM ::= {
          mda-sha1 |
          mda-sha224 |
          mda-sha256 |
          mda-sha384 |
          mda-sha512 |
          mda-sha3-224 |
          mda-sha3-256 |
          mda-sha3-384 |
          mda-sha3-512,
          ... }

      KeyDerivationFunction ::=  AlgorithmIdentifier{ KEY-DERIVATION,
                                     { KeyDevAlgs } }

      KeyDevAlgs KEY-DERIVATION ::= {
          kda-hkdf-with-sha3-224 |
          kda-hkdf-with-sha3-256 |
          kda-hkdf-with-sha3-384 |
          kda-hkdf-with-sha3-512 |
          kda-kmac128 |
          kda-kmac256 |
          kda-kdf2 |
          kda-kdf3,
          ... }

      END
   <CODE ENDS>
```

Author’s Address

Russ Housley
Vigil Security, LLC
Herndon, VA
United States of America
Email: housley@vigilsec.com

Internet Engineering Task Force
Internet-Draft
Obsoletes: 8954 (if approved)
Updates: 6960 (if approved)
Intended status: Standards Track
Expires: 7 November 2024

H. Sharma, Ed.
Netskope Inc

6 May 2024

Online Certificate Status Protocol (OCSP) Nonce Extension
draft-ietf-lamps-ocsp-nonce-update-07

Abstract

RFC 8954 imposed the size constraints on the optional Nonce extension for the Online Certificate Status Protocol (OCSP). OCSP is used for checking the status of a certificate, and the Nonce extension is used to cryptographically bind an OCSP response message to a particular OCSP request message.

Some environments use cryptographic algorithms that generate a Nonce value that is longer than 32 octets. This document updates the maximum allowed length of Nonce to 128 octets. This document also modifies Nonce section to clearly define the encoding format and values distinctively for an easier implementation and understanding. This document obsoletes RFC 8954 and provides updated ASN.1 modules for OCSP, updates RFC 6960.

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 7 November 2024.

Copyright Notice

Table of Contents

1. Introduction

Nonce extension was previously defined in Section 4.4.1 of [RFC6960] and updated in [RFC8954]. [RFC8954] limits the maximum Nonce length to 32 octets. To support cryptographic algorithms that generate a Nonce that is longer than 32 octets, this document updates the maximum allowed size of the Nonce to 128 octets. In addition, this document recommends that the OCSP client and responder use a Nonce with a minimum length of 32 octets.

1.1. 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.

2. OCSP Extensions

The message formats for OCSP requests and responses are defined in [RFC6960] and Nonce extension was updated in [RFC8954]. [RFC6960] also defines the standard extensions for OCSP messages based on the extension model employed in X.509 version 3 certificates (see [RFC5280]). [RFC8954] replaces this section to limit the minimum and maximum length for the Nonce value. This document extends the maximum allowed nonce length to 128 octets and does not change the specifications of any of the other standard extensions defined in [RFC6960].

2.1. Nonce Extension

The Nonce cryptographically binds a request and a response to prevent replay attacks. The Nonce is included as one of the requestExtensions in requests; in responses, it would be included as one of the responseExtensions. In both the request and the response, the Nonce will be identified by the object identifier id-pkix-ocsp-nonce, while the extnValue is the encoded value of Nonce. If the Nonce extension is present, then the length of the Nonce MUST be at least 1 octet and can be up to 128 octets.

An OCSP client that implements this document MUST use a minimum length of 32 octets for Nonce in the Nonce extension. [RFC8954] compliant implementations will be unable to process nonces generated per the new specification with sizes in excess of the limit of 32 octets that was specified in [RFC8954].

An OCSP responder that implements this document MUST reject any OCSP request that has a Nonce with a length of either 0 octets or more than 128 octets, with the malformedRequest OCSPResponseStatus as described in Section 4.2.1 of [RFC6960]. Responders, supporting the Nonce extension, MUST accept Nonce lengths of at least 16 octets and MAY choose to ignore the Nonce extension for requests where the length of the Nonce is less than 16 octets or more than 32 octets.

The value of the Nonce MUST be generated using a cryptographically strong pseudorandom number generator (see [RFC4086]). The minimum Nonce length of 1 octet is defined to provide backward compatibility with older clients that follow [RFC6960].

```
id-pkix-ocsp           OBJECT IDENTIFIER ::= { id-ad-ocsp }
id-pkix-ocsp-nonce     OBJECT IDENTIFIER ::= { id-pkix-ocsp 2 }
Nonce ::= OCTET STRING(SIZE(1..128))
```

Example of an encoded OCSP Nonce extension with 32 octet Nonce in hexadecimal format.

```
30 2f 06 09 2b 06 01 05 05 07 30 01 02 04 22 04
20 dd 49 d4 07 2c 44 9d a1 c3 17 bd 1c 1b df fe
db e1 50 31 2e c4 cd 0a dd 18 e5 bd 6f 84 bf 14
c8
```

Here is the decoded version of the above example.
Offset, Length and Object Identifier are in decimal.

```
Offset  Length
0       47    : SEQUENCE {
2       9     :   OBJECT IDENTIFIER ocspNonce (1 3 6 1 5 5 7 48 1 2)
13      34    :   OCTET STRING, encapsulates {
15      32    :     OCTET STRING
              :       DD 49 D4 07 2C 44 9D A1 C3 17 BD 1C 1B DF FE DB
              :       E1 50 31 2E C4 CD 0A DD 18 E5 BD 6F 84 BF 14 C8
              :     }
              :   }
```

3. Security Considerations

The security considerations of OCSP, in general, are described in [RFC6960]. During the interval in which the previous OCSP response for a certificate is not expired but the responder has a changed status for that certificate, a copy of that OCSP response can be used to indicate that the status of the certificate is still valid. Including a client’s nonce value in the OCSP response makes sure that the response is the latest response from the server and not an old copy.

3.1. Replay Attack

The Nonce extension is used to avoid replay attacks. Since the OCSP responder may choose not to send the Nonce extension in the OCSP response even if the client has sent the Nonce extension in the request [RFC5019], an on-path attacker can intercept the OCSP request and respond with an earlier response from the server without the Nonce extension. This can be mitigated by configuring the server to use a short time interval between the thisUpdate and nextUpdate fields in the OCSP response.

4. IANA Considerations

For the ASN.1 Module in Appendix A.1, IANA is requested to assign an object identifier (OID) for the module identifier to replace TBD1. The OID for the module should be allocated in the "SMI Security for PKIX Module Identifier" registry (1.3.6.1.5.5.7.0), and the Description for the new OID should be set to "id-mod-ocsp-2024-88".

For the ASN.1 Module in Appendix A.2, IANA is requested to assign an object identifier (OID) for the module identifier to replace TBD2. The OID for the module should be allocated in the "SMI Security for PKIX Module Identifier" registry (1.3.6.1.5.5.7.0), and the Description for the new OID should be set to "id-mod-ocsp-2024-08".

Acknowledgements

The authors of this document wish to thank Mohit Sahni for his work to produce [RFC8954].

The authors wish to thank Russ Housley, Corey Bonnell, Michael StJohns and Carl Wallace for the feedback and suggestions.

References

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>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC5019] Deacon, A. and R. Hurst, "The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments", RFC 5019, DOI 10.17487/RFC5019, September 2007, <https://www.rfc-editor.org/info/rfc5019>.

[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>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[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>.

[RFC8954] Sahni, M., Ed., "Online Certificate Status Protocol (OCSP) Nonce Extension", RFC 8954, DOI 10.17487/RFC8954, November 2020, <https://www.rfc-editor.org/info/rfc8954>.

Informative References

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[Errata5891] RFC Errata, Erratum ID 5891, RFC 6960, <https://www.rfc-editor.org/errata/eid5891>.

Appendix A. ASN.1 Modules

This section includes the ASN.1 modules for OCSP and replaces the entirity of Section 5 of [RFC8954]. It addresses Errata id 5891 [Errata5891] as well.

Appendix A.1 includes an ASN.1 module that conforms to the 1998 version of ASN.1 for all syntax elements of OCSP. This module replaces the modules Appendix B.1 of [RFC6960].

Appendix A.2 includes an ASN.1 module, corresponding to the module present in A.1, that conforms to the 2008 version of ASN.1. This module replaces the modules in Section 4 of [RFC5912] and Appendix B.2 of [RFC6960]. Although a 2008 ASN.1 module is provided, the module in Appendix A.1 remains the normative module as per the policy of the PKIX working group.

A.1. OCSP in ASN.1 - 1998 Syntax

```
OCSP-2024-88
      {iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-ocsp-2024-88(TBD1)}

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

IMPORTS

   -- PKIX Certificate Extensions
      AuthorityInfoAccessSyntax, CRLReason, GeneralName
      FROM PKIX1Implicit88 { iso(1) identified-organization(3)
           dod(6) internet(1) security(5) mechanisms(5) pkix(7)
           id-mod(0) id-pkix1-implicit(19) }

      Name, CertificateSerialNumber, Extensions,
      id-kp, id-ad-ocsp, Certificate, AlgorithmIdentifier
      FROM PKIX1Explicit88 { iso(1) identified-organization(3)
           dod(6) internet(1) security(5) mechanisms(5) pkix(7)
           id-mod(0) id-pkix1-explicit(18) };

OCSPRequest ::= SEQUENCE {
   tbsRequest              TBSRequest,
   optionalSignature   [0] EXPLICIT Signature OPTIONAL }

TBSRequest ::= SEQUENCE {
   version             [0] EXPLICIT Version DEFAULT v1,
   requestorName       [1] EXPLICIT GeneralName OPTIONAL,
   requestList             SEQUENCE OF Request,
   requestExtensions   [2] EXPLICIT Extensions OPTIONAL }

Signature ::= SEQUENCE {
   signatureAlgorithm      AlgorithmIdentifier,
   signature               BIT STRING,
   certs               [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

Version ::= INTEGER { v1(0) }

Nonce ::= OCTET STRING(SIZE(1..128))

Request ::= SEQUENCE {
   reqCert                     CertID,
   singleRequestExtensions [0] EXPLICIT Extensions OPTIONAL }

CertID ::= SEQUENCE {
   hashAlgorithm           AlgorithmIdentifier,
```

```
   issuerNameHash          OCTET STRING, -- Hash of issuer's DN
   issuerKeyHash           OCTET STRING, -- Hash of issuer's public key
   serialNumber            CertificateSerialNumber }

OCSPResponse ::= SEQUENCE {
   responseStatus          OCSPResponseStatus,
   responseBytes       [0] EXPLICIT ResponseBytes OPTIONAL }

OCSPResponseStatus ::= ENUMERATED {
   successful          (0),  -- Response has valid confirmations
   malformedRequest    (1),  -- Illegal confirmation request
   internalError       (2),  -- Internal error in issuer
   tryLater            (3),  -- Try again later
                             -- (4) is not used
   sigRequired         (5),  -- Must sign the request
   unauthorized        (6)   -- Request unauthorized
}

ResponseBytes ::= SEQUENCE {
   responseType            OBJECT IDENTIFIER,
   response                OCTET STRING }

BasicOCSPResponse ::= SEQUENCE {
  tbsResponseData          ResponseData,
  signatureAlgorithm       AlgorithmIdentifier,
  signature                BIT STRING,
  certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

ResponseData ::= SEQUENCE {
   version             [0] EXPLICIT Version DEFAULT v1,
   responderID             ResponderID,
   producedAt              GeneralizedTime, -- The format for
                                            -- GeneralizedTime is as
                                            -- specified in Section
                                            -- 4.1.2.5.2 of [RFC5280]
   responses               SEQUENCE OF SingleResponse,
   responseExtensions  [1] EXPLICIT Extensions OPTIONAL }

ResponderID ::= CHOICE {
   byName              [1] Name,
   byKey               [2] KeyHash }

KeyHash ::= OCTET STRING -- SHA-1 hash of responder's public key
                         -- (i.e., the SHA-1 hash of the value of the
                         -- BIT STRING subjectPublicKey [excluding
                         -- the tag, length, and number of unused
                         -- bits] in the responder's certificate)
```

```
SingleResponse ::= SEQUENCE {
   certID                  CertID,
   certStatus              CertStatus,
   thisUpdate              GeneralizedTime,
   nextUpdate          [0] EXPLICIT GeneralizedTime OPTIONAL,
   singleExtensions    [1] EXPLICIT Extensions OPTIONAL }

CertStatus ::= CHOICE {
   good                [0] IMPLICIT NULL,
   revoked             [1] IMPLICIT RevokedInfo,
   unknown             [2] IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
   revocationTime          GeneralizedTime,
   revocationReason    [0] EXPLICIT CRLReason OPTIONAL }

UnknownInfo ::= NULL

ArchiveCutoff ::= GeneralizedTime

AcceptableResponses ::= SEQUENCE OF OBJECT IDENTIFIER

ServiceLocator ::= SEQUENCE {
   issuer                  Name,
   locator                 AuthorityInfoAccessSyntax }

CrlID ::= SEQUENCE {
    crlUrl               [0]     EXPLICIT IA5String OPTIONAL,
    crlNum               [1]     EXPLICIT INTEGER OPTIONAL,
    crlTime              [2]     EXPLICIT GeneralizedTime OPTIONAL }

PreferredSignatureAlgorithms ::= SEQUENCE OF PreferredSignatureAlgorithm

PreferredSignatureAlgorithm ::= SEQUENCE {
   sigIdentifier   AlgorithmIdentifier,
   certIdentifier  AlgorithmIdentifier OPTIONAL }



-- Object Identifiers

id-kp-OCSPSigning            OBJECT IDENTIFIER ::= { id-kp 9 }
id-pkix-ocsp                 OBJECT IDENTIFIER ::= { id-ad-ocsp }
id-pkix-ocsp-basic           OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }
id-pkix-ocsp-nonce           OBJECT IDENTIFIER ::= { id-pkix-ocsp 2 }
id-pkix-ocsp-crl             OBJECT IDENTIFIER ::= { id-pkix-ocsp 3 }
id-pkix-ocsp-response        OBJECT IDENTIFIER ::= { id-pkix-ocsp 4 }
id-pkix-ocsp-nocheck         OBJECT IDENTIFIER ::= { id-pkix-ocsp 5 }
```

```
id-pkix-ocsp-archive-cutoff  OBJECT IDENTIFIER ::= { id-pkix-ocsp 6 }
id-pkix-ocsp-service-locator OBJECT IDENTIFIER ::= { id-pkix-ocsp 7 }
id-pkix-ocsp-pref-sig-algs   OBJECT IDENTIFIER ::= { id-pkix-ocsp 8 }
id-pkix-ocsp-extended-revoke OBJECT IDENTIFIER ::= { id-pkix-ocsp 9 }

END
```

A.2. OCSP in ASN.1 - 2008 Syntax

```
OCSP-2024-08
     {iso(1) identified-organization(3) dod(6) internet(1)
     security(5) mechanisms(5) pkix(7) id-mod(0)
     id-mod-ocsp-2024-08(TBD2)}

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

IMPORTS

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

AlgorithmIdentifier{}, DIGEST-ALGORITHM, SIGNATURE-ALGORITHM, PUBLIC-KEY
FROM AlgorithmInformation-2009 -- From [RFC5912]
   {iso(1) identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7) id-mod(0)
   id-mod-algorithmInformation-02(58)}

AuthorityInfoAccessSyntax, GeneralName, CrlEntryExtensions, CRLReason
FROM PKIX1Implicit-2009 -- From [RFC5912]
   {iso(1) identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-implicit-02(59)}

Name, CertificateSerialNumber, id-kp, id-ad-ocsp, Certificate
FROM PKIX1Explicit-2009 -- From [RFC5912]
   {iso(1) identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-explicit-02(51)}

sa-dsaWithSHA1, sa-rsaWithMD2, sa-rsaWithMD5, sa-rsaWithSHA1
FROM PKIXAlgs-2009 -- From [RFC5912]
   {iso(1) identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) pkix(7) id-mod(0)
   id-mod-pkix1-algorithms2008-02(56)};
```

```
OCSPRequest     ::=     SEQUENCE {
   tbsRequest                  TBSRequest,
   optionalSignature   [0]     EXPLICIT Signature OPTIONAL }

TBSRequest      ::=     SEQUENCE {
   version             [0] EXPLICIT Version DEFAULT v1,
   requestorName       [1] EXPLICIT GeneralName OPTIONAL,
   requestList             SEQUENCE OF Request,
   requestExtensions   [2] EXPLICIT Extensions {{re-ocsp-nonce |
                     re-ocsp-response, ...,
                     re-ocsp-preferred-signature-algorithms}} OPTIONAL }

Signature       ::=     SEQUENCE {
   signatureAlgorithm   AlgorithmIdentifier
                            { SIGNATURE-ALGORITHM, {...}},
   signature            BIT STRING,
   certs             [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

Version  ::=  INTEGER  {  v1(0) }

Nonce ::= OCTET STRING(SIZE(1..128))

Request ::=     SEQUENCE {
   reqCert                     CertID,
   singleRequestExtensions     [0] EXPLICIT Extensions
                                       { {re-ocsp-service-locator,
                                              ...}} OPTIONAL }

CertID ::= SEQUENCE {
   hashAlgorithm            AlgorithmIdentifier
                                {DIGEST-ALGORITHM, {...}},
   issuerNameHash     OCTET STRING, -- Hash of issuer's DN
   issuerKeyHash      OCTET STRING, -- Hash of issuer's public key
   serialNumber       CertificateSerialNumber }

OCSPResponse ::= SEQUENCE {
  responseStatus         OCSPResponseStatus,
  responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL }



OCSPResponseStatus ::= ENUMERATED {
   successful            (0), -- Response has valid confirmations
   malformedRequest      (1), -- Illegal confirmation request
   internalError         (2), -- Internal error in issuer
   tryLater              (3), -- Try again later
                              -- (4) is not used
   sigRequired           (5), -- Must sign the request
```

```
   unauthorized          (6)  -- Request unauthorized
}

RESPONSE ::= TYPE-IDENTIFIER

ResponseSet RESPONSE ::= {basicResponse, ...}

ResponseBytes ::=       SEQUENCE {
   responseType        RESPONSE.
                           &id ({ResponseSet}),
   response            OCTET STRING (CONTAINING RESPONSE.
                           &Type({ResponseSet}{@responseType}))}

basicResponse RESPONSE ::=
   { BasicOCSPResponse IDENTIFIED BY id-pkix-ocsp-basic }

BasicOCSPResponse       ::= SEQUENCE {
  tbsResponseData      ResponseData,
  signatureAlgorithm   AlgorithmIdentifier{SIGNATURE-ALGORITHM,
                           {sa-dsaWithSHA1 | sa-rsaWithSHA1 |
                                sa-rsaWithMD5 | sa-rsaWithMD2, ...}},
  signature            BIT STRING,
  certs            [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

ResponseData ::= SEQUENCE {
  version              [0] EXPLICIT Version DEFAULT v1,
  responderID              ResponderID,
  producedAt               GeneralizedTime,
  responses                SEQUENCE OF SingleResponse,
  responseExtensions   [1] EXPLICIT Extensions
                              {{re-ocsp-nonce, ...,
                                re-ocsp-extended-revoke}} OPTIONAL }

ResponderID ::= CHOICE {
  byName   [1] Name,
  byKey    [2] KeyHash }

KeyHash ::= OCTET STRING -- SHA-1 hash of responder's public key
                         -- (excluding the tag and length fields)

SingleResponse ::= SEQUENCE {
  certID                       CertID,
  certStatus                   CertStatus,
  thisUpdate                   GeneralizedTime,
  nextUpdate           [0]     EXPLICIT GeneralizedTime OPTIONAL,
  singleExtensions     [1]     EXPLICIT Extensions{{re-ocsp-crl |
                                           re-ocsp-archive-cutoff |
                                           CrlEntryExtensions, ...}
```

```
                                          } OPTIONAL }

CertStatus ::= CHOICE {
   good                [0]     IMPLICIT NULL,
   revoked             [1]     IMPLICIT RevokedInfo,
   unknown             [2]     IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
   revocationTime              GeneralizedTime,
   revocationReason    [0]     EXPLICIT CRLReason OPTIONAL }

UnknownInfo ::= NULL

ArchiveCutoff ::= GeneralizedTime

AcceptableResponses ::= SEQUENCE OF RESPONSE.&id({ResponseSet})

ServiceLocator ::= SEQUENCE {
   issuer    Name,
   locator   AuthorityInfoAccessSyntax }

CrlID ::= SEQUENCE {
   crlUrl               [0]     EXPLICIT IA5String OPTIONAL,
   crlNum               [1]     EXPLICIT INTEGER OPTIONAL,
   crlTime              [2]     EXPLICIT GeneralizedTime OPTIONAL }

PreferredSignatureAlgorithms ::= SEQUENCE OF PreferredSignatureAlgorithm

PreferredSignatureAlgorithm ::= SEQUENCE {
  sigIdentifier  AlgorithmIdentifier{SIGNATURE-ALGORITHM, {...}},
  certIdentifier AlgorithmIdentifier{PUBLIC-KEY, {...}} OPTIONAL
}

-- Certificate Extensions

ext-ocsp-nocheck EXTENSION ::= { SYNTAX NULL IDENTIFIED
                                 BY id-pkix-ocsp-nocheck }

-- Request Extensions

re-ocsp-nonce EXTENSION ::= { SYNTAX Nonce
                              IDENTIFIED BY id-pkix-ocsp-nonce }

re-ocsp-response EXTENSION ::= { SYNTAX AcceptableResponses IDENTIFIED
                                 BY id-pkix-ocsp-response }

re-ocsp-service-locator EXTENSION ::= { SYNTAX ServiceLocator
                                        IDENTIFIED BY
```

```
                                        id-pkix-ocsp-service-locator }

re-ocsp-preferred-signature-algorithms EXTENSION ::= {
  SYNTAX PreferredSignatureAlgorithms
  IDENTIFIED BY id-pkix-ocsp-pref-sig-algs  }

-- Response Extensions

re-ocsp-crl EXTENSION ::= { SYNTAX CrlID IDENTIFIED BY
                               id-pkix-ocsp-crl }

re-ocsp-archive-cutoff EXTENSION ::= { SYNTAX ArchiveCutoff
                                       IDENTIFIED BY
                                       id-pkix-ocsp-archive-cutoff }

re-ocsp-extended-revoke EXTENSION ::= { SYNTAX NULL IDENTIFIED BY
                                        id-pkix-ocsp-extended-revoke }

-- Object Identifiers

id-kp-OCSPSigning            OBJECT IDENTIFIER ::= { id-kp 9 }
id-pkix-ocsp                 OBJECT IDENTIFIER ::= id-ad-ocsp
id-pkix-ocsp-basic           OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }
id-pkix-ocsp-nonce           OBJECT IDENTIFIER ::= { id-pkix-ocsp 2 }
id-pkix-ocsp-crl             OBJECT IDENTIFIER ::= { id-pkix-ocsp 3 }
id-pkix-ocsp-response        OBJECT IDENTIFIER ::= { id-pkix-ocsp 4 }
id-pkix-ocsp-nocheck         OBJECT IDENTIFIER ::= { id-pkix-ocsp 5 }
id-pkix-ocsp-archive-cutoff  OBJECT IDENTIFIER ::= { id-pkix-ocsp 6 }
id-pkix-ocsp-service-locator OBJECT IDENTIFIER ::= { id-pkix-ocsp 7 }
id-pkix-ocsp-pref-sig-algs   OBJECT IDENTIFIER ::= { id-pkix-ocsp 8 }
id-pkix-ocsp-extended-revoke OBJECT IDENTIFIER ::= { id-pkix-ocsp 9 }

END
```

Author's Address

Himanshu Sharma (editor)
Netskope Inc
2445 Augustine Dr 3rd floor
Santa Clara, California 95054
United States of America
Email: himanshu@netskope.com
URI: www.netskope.com

Limited Additional Mechanisms for PKIX and SMIME
Internet-Draft
Obsoletes: 5990 (if approved)
Intended status: Standards Track
Expires: 1 November 2024

R. Housley
Vigil Security
S. Turner
sn3rd

30 April 2024

Use of the RSA-KEM Algorithm in the Cryptographic Message Syntax (CMS) draft-ietf-lamps-rfc5990bis-06

Abstract

The RSA Key Encapsulation Mechanism (RSA-KEM) Algorithm is a one-pass (store-and-forward) cryptographic mechanism for an originator to securely send keying material to a recipient using the recipient's RSA public key. The RSA-KEM Algorithm is specified in Clause 11.5 of ISO/IEC: 18033-2:2006. This document specifies the conventions for using the RSA-KEM Algorithm as a standalone KEM algorithm and the conventions for using the RSA-KEM Algorithm with the Cryptographic Message Syntax (CMS) using KEMRecipientInfo as specified in RFC XXXX. This document obsoletes RFC 5990.

RFC EDITOR: Please replace XXXX with the RFC number assigned to draft-ietf-lamps-cms-kemri.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-lamps-rfc5990bis/.

Discussion of this document takes place on the Limited Additional Mechanisms for PKIX and SMIME Working Group mailing list (mailto:spasm@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/spasm/. Subscribe at https://www.ietf.org/mailman/listinfo/spasm/.

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/.

Table of Contents

1. Introduction

The RSA Key Encapsulation Mechanism (RSA-KEM) Algorithm is a one-pass (store-and-forward) cryptographic mechanism for an originator to securely send keying material to a recipient using the recipient's RSA public key. The RSA-KEM Algorithm is specified in Clause 11.5 of [ISO18033-2].

The RSA-KEM Algorithm takes a different approach than other RSA key transport mechanisms [RFC8017], with the goal of providing higher security assurance while also satisfying the KEM interface. The RSA-KEM Algorithm encrypts a random integer with the recipient's RSA public key, and derives a shared secret from the random integer. The originator and recipient can derive a symmetric key from the shared secret. For example, a key-encryption key can be derived from the shared secret to wrap a content-encryption key.

In the Cryptographic Message Syntax (CMS) [RFC5652] using KEMRecipientInfo [I-D.ietf-lamps-cms-kemri], the shared secret value is input to a key-derivation function to compute a key-encryption key, and wrap a symmetric content-encryption key with the key-encryption key. In this way, the originator and the recipient end up with the same content-encryption key.

For completeness, a specification of the RSA-KEM Algorithm is given in Appendix A of this document; ASN.1 syntax is given in Appendix B.

1.1. RSA-KEM Algorithm Rationale

The RSA-KEM Algorithm provides higher security assurance than other variants of the RSA cryptosystem for two reasons. First, the input to the underlying RSA operation is a string-encoded random integer between 0 and n-1, where n is the RSA modulus, so it does not have any structure that could be exploited by an adversary. Second, the input is independent of the keying material so the result of the RSA decryption operation is not directly available to an adversary. As a result, the RSA-KEM Algorithm enjoys a "tight" security proof in the random oracle model. (In other padding schemes, such as PKCS #1 v1.5 [RFC8017], the input has structure and/or depends on the keying material, and the provable security assurances are not as strong.)

The approach is also architecturally convenient because the public-key operations are separate from the symmetric operations on the keying material. Another benefit is that the length of the keying material is determined by the symmetric algorithms, not the size of the RSA modulus.

1.2. RSA-KEM Algorithm Summary

All KEM algorithms provide three functions: KeyGen(), Encapsulate(), and Decapsulate().

The following summarizes these three functions for RSA-KEM:

KeyGen() -> (pk, sk):

Generate the public key (pk) and a private key (sk) as described in Section 3 of [RFC8017].

Encapsulate(pk) -> (ct, SS):

Given the recipient's public key (pk), produce a ciphertext (ct) to be passed to the recipient and a shared secret (SS) for use by the originator, as follows:

1. Generate a random integer z between 0 and n-1.

2. Encrypt the integer z with the recipient's RSA public key to obtain the ciphertext:

    ct = z^e mod n

3. Derive a shared secret from the integer z using a Key Derivation Function (KDF):

    SS = KDF(z)

4. The ciphertext and the shared secret are returned by the function. The originator sends the ciphertext to the recipient.

Decapsulate(sk, ct) -> SS:

Given the private key (sk) and the ciphertext (ct), produce the shared secret (SS) for the recipient as follows:

1. Decrypt the ciphertext with the recipient's RSA private key to obtain the random integer z:

    z = ct^d mod n

2. Derive a shared secret from the integer z:

    SS = KDF(z)

3. The shared secret is returned by the function.

1.3. CMS KEMRecipientInfo Processing Summary

To support the RSA-KEM algorithm, the CMS originator MUST implement Encapsulate().

Given a content-encryption key CEK, the RSA-KEM Algorithm processing by the originator to produce the values that are carried in the CMS KEMRecipientInfo can be summarized as:

1. Obtain the shared secret using the Encapsulate() function of the RSA-KEM algorithm and the recipient's RSA public key:

    (ct, SS) = Encapsulate(pk)

2. Derive a key-encryption key KEK from the shared secret:

    KEK = KDF(SS)

3. Wrap the CEK with the KEK to obtain wrapped keying material WK:

    WK = WRAP(KEK, CEK)

4. The originator sends the ciphertext and WK to the recipient in the CMS KEMRecipientInfo structure.

To support the RSA-KEM algorithm, the CMS recipient MUST implement Decapsulate().

The RSA-KEM algorithm recipient processing of the values obtained from the KEMRecipientInfo structure can be summarized as:

1. Obtain the shared secret using the Decapsulate() function of the RSA-KEM algorithm and the recipient's RSA private key:

    SS = Decapsulate(sk, ct)

2. Derive a key-encryption key KEK from the shared secret:

    KEK = KDF(SS)

3. Unwrap the WK with the KEK to obtain content-encryption key CEK:

```
CEK = UNWRAP(KEK, WK)
```

Note that the KDF used to process the KEMRecipientInfo structure MAY be different from the KDF used to derive the shared secret in the RSA-KEM algorithm.

1.4. Conventions and Definitions

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

1.5. ASN.1

CMS values are generated using ASN.1 [X.680], which uses the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X.690].

1.6. Changes Since RFC 5990

RFC 5990 [RFC5990] specified the conventions for using the RSA-KEM Algorithm in CMS as a key transport algorithm. That is, it used KeyTransRecipientInfo [RFC5652] for each recipient. Since the publication of RFC 5990, a new KEMRecipientInfo structure [I-D.ietf-lamps-cms-kemri] has been defined to support KEM algorithms. When the id-rsa-kem algorithm identifier appears in the SubjectPublicKeyInfo field of a certificate, the complex parameter structure defined in RFC 5990 can be omitted; however, the parameters are allowed for backward compatibility. Also, to avoid visual confusion with id-kem-rsa, id-rsa-kem-spki is introduced as an alias for id-rsa-kem.

RFC 5990 used EK as the EncryptedKey, which is the concatenation of the ciphertext C and the wrapped key WK, EK = (C || WK). The use of EK was necessary to align with the KeyTransRecipientInfo structure. In this document, the ciphertext and the wrapped key are sent in separate fields of the KEMRecipientInfo structure. In particular, the ciphertext is carried in the kemct field, and wrapped key is carried in the encryptedKey field. See Appendix A for details about the computation of the ciphertext.

RFC 5990 included support for Camellia and Triple-DES block ciphers; discussion of these block ciphers is removed from this document, but the algorithm identifiers remain in the ASN.1 Module Appendix B.2.

RFC 5990 included support for SHA-1 hash function; discussion of this hash function is removed from this document, but the algorithm identifier remains in the ASN.1 module Appendix B.2.

RFC 5990 required support for the KDF3 key-derivation function [ANS-X9.44]; this document continues to require support for the KDF3 key-derivation function, but it requires support for SHA-256 [SHS] as the hash function.

RFC 5990 recommended support for alternatives to KDF3 and AES-Wrap-128; this document simply states that other key-derivation functions and other key-encryption algorithms MAY be supported.

RFC 5990 supported the future definition of additional KEM algorithms that use RSA; this document supports only one, and it is identified by the id-kem-rsa object identifier.

RFC 5990 included an ASN.1 module; this document provides an alternative ASN.1 module that follows the conventions established in [RFC5911], [RFC5912], and [RFC6268]. The new ASN.1 module Appendix B.2 produces the same bits-on-the-wire as the one in RFC 5990.

2. Use of the RSA-KEM Algorithm in CMS

The RSA-KEM Algorithm MAY be employed for one or more recipients in the CMS enveloped-data content type [RFC5652], the CMS authenticated-data content type [RFC5652], or the CMS authenticated-enveloped-data content type [RFC5083]. In each case, the KEMRecipientInfo [I-D.ietf-lamps-cms-kemri] is used with the RSA-KEM Algorithm to securely transfer the content-encryption key from the originator to the recipient.

2.1. Underlying Components

A CMS implementation that supports the RSA-KEM Algorithm MUST support at least the following underlying components:

* For the key-derivation function, an implementation MUST support KDF3 [ANS-X9.44] with SHA-256 [SHS].

* For key-wrapping, an implementation MUST support the AES-Wrap-128 [RFC3394] key-encryption algorithm.

An implementation MAY also support other key-derivation functions and other key-encryption algorithms as well.

2.2. RecipientInfo Conventions

When the RSA-KEM Algorithm is employed for a recipient, the RecipientInfo alternative for that recipient MUST be OtherRecipientInfo using the KEMRecipientInfo structure [I-D.ietf-lamps-cms-kemri]. The fields of the KEMRecipientInfo MUST have the following values:

version is the syntax version number; it MUST be 0.

rid identifies the recipient's certificate or public key.

kem identifies the KEM algorithm; it MUST contain id-kem-rsa.

kemct is the ciphertext produced for this recipient; it contains C from steps 1 and 2 of Originator's Operations in Appendix A.

kdf identifies the key-derivation function (KDF). Note that the KDF used for CMS RecipientInfo process MAY be different than the KDF used within the RSA-KEM Algorithm.

kekLength is the size of the key-encryption key in octets.

ukm is an optional random input to the key-derivation function.

wrap identifies a key-encryption algorithm used to encrypt the keying material.

encryptedKey is the result of encrypting the keying material with the key-encryption key. When used with the CMS enveloped-data content type [RFC5652], the keying material is a content-encryption key. When used with the CMS authenticated-data content type [RFC5652], the keying material is a message-authentication key. When used with the CMS authenticated-enveloped-data content type [RFC5083], the keying material is a content-authenticated-encryption key.

NOTE: For backward compatibility, implementations MAY also support RSA-KEM Key Transport Algorithm, identified by id-rsa-kem-spki, which uses KeyTransRecipientInfo as specified in [RFC5990].

2.3. Certificate Conventions

The conventions specified in this section augment RFC 5280 [RFC5280].

A recipient who employs the RSA-KEM Algorithm MAY identify the public key in a certificate by the same AlgorithmIdentifier as for the PKCS #1 v1.5 algorithm, that is, using the rsaEncryption object identifier [RFC8017]. The fact that the recipient will accept RSA-KEM with this public key is not indicated by the use of this object identifier. The willingness to accept the RSA-KEM Algorithm MAY be signaled by the use of the SMIMECapabilities Attribute as specified in Section 2.5.2. of [RFC8551] or the SMIMECapabilities certificate extension as specified in [RFC4262].

If the recipient wishes only to employ the RSA-KEM Algorithm with a given public key, the recipient MUST identify the public key in the certificate using the id-rsa-kem-spki object identifier; see Appendix B. The use of the id-rsa-kem-spki object identifier allows certificates that were issued to be compatible with RSA-KEM Key Transport to also be used with this specification. When the id-rsa-kem-spki object identifier appears in the SubjectPublicKeyInfo algorithm field of the certificate, the parameters field from AlgorithmIdentifier SHOULD be absent. That is, the AlgorithmIdentifier SHOULD be a SEQUENCE of one component, the id-rsa-kem-spki object identifier. With absent parameters, the KDF3 key-derivation function [ANS-X9.44] with SHA-256 [SHS] are used to derive the shared secret.

When the AlgorithmIdentifier parameters are present, the GenericHybridParameters MUST be used. Within the kem element, the algorithm identifier MUST be set to id-kem-rsa, and RsaKemParameters MUST be included. As described in Section 2.4, the GenericHybridParameters constrain the values that can be used with the RSA public key for the kdf, kekLength, and wrap fields of the KEMRecipientInfo structure.

Regardless of the AlgorithmIdentifier used, the RSA public key MUST be carried in the subjectPublicKey BIT STRING within the SubjectPublicKeyInfo field of the certificate using the RSAPublicKey type defined in [RFC8017].

The intended application for the public key MAY be indicated in the key usage certificate extension as specified in Section 4.2.1.3 of [RFC5280]. If the keyUsage extension is present in a certificate that conveys an RSA public key with the id-rsa-kem-spki object identifier as discussed above, then the key usage extension MUST contain only the following value:

keyEncipherment

The digitalSignature and dataEncipherment values SHOULD NOT be present. That is, a public key intended to be employed only with the RSA-KEM Algorithm SHOULD NOT also be employed for data encryption or for digital signatures. Good cryptographic practice employs a given RSA key pair in only one scheme. This practice avoids the risk that vulnerability in one scheme may compromise the security of the other, and may be essential to maintain provable security.

2.4. SMIMECapabilities Attribute Conventions

Section 2.5.2 of [RFC8551] defines the SMIMECapabilities attribute to announce a partial list of algorithms that an S/MIME implementation can support. When constructing a CMS signed-data content type [RFC5652], a compliant implementation MAY include the SMIMECapabilities attribute that announces support for the RSA-KEM Algorithm.

The SMIMECapability SEQUENCE representing the RSA-KEM Algorithm MUST include the id-rsa-kem-spki object identifier in the capabilityID field; see Appendix B for the object identifier value, and see Appendix C for examples. When the id-rsa-kem-spki object identifier appears in the capabilityID field and the parameters are present, then the parameters field MUST use the GenericHybridParameters type.

```
GenericHybridParameters ::= SEQUENCE {
  kem  KeyEncapsulationMechanism,
  dem  DataEncapsulationMechanism }
```

The fields of the GenericHybridParameters type have the following meanings:

kem is an AlgorithmIdentifer. The algorithm field MUST be set to id-kem-rsa, and the parameters field MUST be RsaKemParameters, which is a SEQUENCE of an AlgorithmIdentifier that identifies the supported key-derivation function and a positive INTEGER that identifies the length of the key-encryption key in octets.

dem is an AlgorithmIdentifier. The algorithm field MUST be present, and it identifies the key-encryption algorithm. The parameters are optional. If the GenericHybridParameters are present, then the provided dem value MUST be used in the wrap field of KEMRecipientInfo.

If the GenericHybridParameters are present, then the provided kem value MUST be used as the key-derivation function in the kdf field of KEMRecipientInfo, and the provided key length MUST be used in the kekLength of KEMRecipientInfo.

3. Security Considerations

The RSA-KEM Algorithm should be considered as a replacement for the widely implemented PKCS #1 v1.5 [RFC8017] for new applications that use CMS to avoid potential vulnerabilities to chosen-ciphertext attacks and gain a tighter security proof; however, the RSA-KEM Algorithm has the disadvantage of slightly longer encrypted keying material. With PKCS #1 v1.5, the originator encrypts the key-encryption key directly with the recipient's RSA public key. With the RSA-KEM, the key-encryption key is encrypted separately.

The security of the RSA-KEM Algorithm can be shown to be tightly related to the difficulty of either solving the RSA problem, or breaking the underlying symmetric key-encryption algorithm, if the underlying key-derivation function is modeled as a random oracle, and assuming that the symmetric key-encryption algorithm satisfies the properties of a data encapsulation mechanism [SHOUP]. While in practice a random-oracle result does not provide an actual security proof for any particular key-derivation function, the result does provide assurance that the general construction is reasonable; a key-derivation function would need to be particularly weak to lead to an attack that is not possible in the random-oracle model.

The RSA key size and the underlying components need to be selected consistent with the desired security level. Several security levels have been identified in the NIST SP 800-57 Part 1 [NISTSP800-57pt1r5]. To achieve 128-bit security, the RSA key size SHOULD be at least 3072 bits, the key-derivation function SHOULD make use of SHA-256, and the symmetric key-encryption algorithm SHOULD be AES Key Wrap with a 128-bit key.

Implementations MUST protect the RSA private key, the key-encryption key, the content-encryption key, message-authentication key, and the content-authenticated-encryption key. Disclosure of the RSA private key could result in the compromise of all messages protected with that key. Disclosure of the key-encryption key, the content-encryption key, or the content-authenticated-encryption key could result in compromise of the associated encrypted content. Disclosure of the key-encryption key, the message-authentication key, or the content-authenticated-encryption key could allow modification of the associated authenticated content.

Additional considerations related to key management may be found in [NISTSP800-57pt1r5].

The security of the RSA-KEM Algorithm depends on a quality random number generator. For further discussion on random number generation, see [RFC4086].

The RSA-KEM Algorithm does not use an explicit padding scheme; instead, an encoded random value (z) between zero and the RSA modulus minus one (n-1) is directly encrypted with the recipient's RSA public key. The IntegerToString(z, nLen) encoding produces a string that is the full length of the RSA modulus. In addition, the random value is passed through a key-derivation function (KDF) to reduce possible harm from a poorly implemented random number source or a maliciously chosen random value (z). Implementations SHOULD NOT use z directly for any purpose.

As long as a fresh random integer z is chosen as part of each invocation of the Encapsulate() function, RSA-KEM does not degrade as the number of ciphertexts increases. Since RSA encryption provides a bijective map, a collision in the KDF is the only way that RSA-KEM can produce more than one ciphertext that encapsulates the same shared secret.

The RSA-KEM Algorithm provides a fixed-length ciphertext. The recipient MUST check that the received byte string is the expected length and the length corresponds to an integer in the expected range prior to attempting decryption with their RSA private key as described in Steps 1 and 2 of Appendix A.2.

Implementations SHOULD NOT reveal information about intermediate values or calculations, whether by timing or other "side channels", otherwise an opponent may be able to determine information about the keying data and/or the recipient's private key. Although not all intermediate information may be useful to an opponent, it is preferable to conceal as much information as is practical, unless analysis specifically indicates that the information would not be useful to an opponent.

Generally, good cryptographic practice employs a given RSA key pair in only one scheme. This practice avoids the risk that vulnerability in one scheme may compromise the security of the other, and may be essential to maintain provable security. While RSA public keys have often been employed for multiple purposes such as key transport and digital signature without any known bad interactions, for increased security assurance, such combined use of an RSA key pair is NOT RECOMMENDED in the future (unless the different schemes are specifically designed to be used together).

Accordingly, an RSA key pair used for the RSA-KEM Algorithm SHOULD NOT also be used for digital signatures. Indeed, the Accredited Standards Committee X9 (ASC X9) requires such a separation between key pairs used for key establishment and key pairs used for digital signature [ANS-X9.44]. Continuing this principle of key separation, a key pair used for the RSA-KEM Algorithm SHOULD NOT be used with other key establishment schemes, or for data encryption, or with more than one set of underlying algorithm components.

It is acceptable to use the same RSA key pair for RSA-KEM Key Transport as specified in [RFC5990] and this specification. This is acceptable because the operations involving the RSA public key and the RSA private key are identical in the two specifications.

Parties MAY gain assurance that implementations are correct through formal implementation validation, such as the NIST Cryptographic Module Validation Program (CMVP) [CMVP].

4. IANA Considerations

For the ASN.1 Module in Appendix B.2, IANA is requested to assign an object identifier (OID) for the module identifier. The OID for the module should be allocated in the "SMI Security for S/MIME Module Identifier" registry (1.2.840.113549.1.9.16.0), and the Description for the new OID should be set to "id-mod-cms-rsa-kem-2023".

5. References

5.1. Normative References

[ANS-X9.44]
American National Standards Institute, "Public Key Cryptography for the Financial Services Industry -- Key Establishment Using Integer Factorization Cryptography", American National Standard X9.44, 2007.

[I-D.ietf-lamps-cms-kemri]
Housley, R., Gray, J., and T. Okubo, "Using Key Encapsulation Mechanism (KEM) Algorithms in the Cryptographic Message Syntax (CMS)", Work in Progress, Internet-Draft, draft-ietf-lamps-cms-kemri-08, 6 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-cms-kemri-08>.

[ISO18033-2]
ISO/IEC JTC 1/SC 27, "Information technology -- Security techniques -- Encryption algorithms -- Part 2: Asymmetric ciphers", ISO/IEC 18033-2:2006, 2006, <https://www.iso.org/standard/37971.html>.

[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>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <https://www.rfc-editor.org/info/rfc3394>.

[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, DOI 10.17487/RFC5083, November 2007, <https://www.rfc-editor.org/info/rfc5083>.

[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>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5911] Hoffman, P. and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, DOI 10.17487/RFC5911, June 2010, <https://www.rfc-editor.org/info/rfc5911>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[RFC6268] Schaad, J. and S. Turner, "Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX)", RFC 6268, DOI 10.17487/RFC6268, July 2011, <https://www.rfc-editor.org/info/rfc6268>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[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>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/ Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard", DOI 10.6028/nist.fips.180-4, July 2015, <https://doi.org/10.6028/nist.fips.180-4>.

[X.680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

[X.690] ITU-T, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

5.2. Informative References

[CMVP] National Institute of Standards and Technology, "Cryptographic Module Validation Program", 2016, <https://csrc.nist.gov/projects/cryptographic-module-validation-program>.

[NISTSP800-57pt1r5] National Institute of Standards and Technology, "Recommendation for Key Management:Part 1 - General", DOI 10.6028/nist.sp.800-57pt1r5, May 2020, <https://doi.org/10.6028/nist.sp.800-57pt1r5>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4262] Santesson, S., "X.509 Certificate Extension for Secure/ Multipurpose Internet Mail Extensions (S/MIME) Capabilities", RFC 4262, DOI 10.17487/RFC4262, December 2005, <https://www.rfc-editor.org/info/rfc4262>.

[RFC5990] Randall, J., Kaliski, B., Brainard, J., and S. Turner, "Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS)", RFC 5990, DOI 10.17487/RFC5990, September 2010, <https://www.rfc-editor.org/info/rfc5990>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/info/rfc6194>.

[SHOUP] Shoup, V., "A Proposal for an ISO Standard for Public Key Encryption", Cryptology ePrint Archive Paper 2001/112, 2001, <https://eprint.iacr.org/2001/112>.

Appendix A. RSA-KEM Algorithm

The RSA-KEM Algorithm is a one-pass (store-and-forward) cryptographic mechanism for an originator to securely send keying material to a recipient using the recipient's RSA public key.

With the RSA-KEM Algorithm, an originator encrypts a random integer (z) with the recipient's RSA public key to produce a ciphertext (ct), and the originator derives a shared secret (SS) from the random integer (z). The originator then sends the ciphertext (ct) to the recipient. The recipient decrypts the ciphertext (ct) using their private key to recover the random integer (z), and the recipient derives a shared secret (SS) from the random integer(z). In this way, originator and recipient obtain the same shared secret (SS).

The RSA-KEM Algorithm depends on a key-derivation function (KDF), which is used to derive the shared secret (SS). Many key-derivation functions support the inclusion of other information in addition to the shared secret (SS) in the input to the function; however, no other information is included as an input to the KDF by the RSA-KEM Algorithm.

A.1. Originator's Operations: RSA-KEM Encapsulate()

Let (n,e) be the recipient's RSA public key; see [RFC8017] for details.

Let nLen denote the length in bytes of the modulus n, i.e., the least integer such that 2^(8*nLen) > n.

The originator performs the following operations:

1. Generate a random integer z between 0 and n-1 (see note), and convert z to a byte string Z of length nLen, most significant byte first:

    z = RandomInteger (0, n-1)

    Z = IntegerToString (z, nLen)

2. Encrypt the random integer z using the recipient's RSA public key (n,e), and convert the resulting integer c to a ciphertext C, a byte string of length nLen:

    c = z^e mod n

    ct = IntegerToString (c, nLen)

3. Derive a symmetric shared secret SS of length ssLen bytes from the byte string Z using the underlying key-derivation function:

    SS = KDF (Z, ssLen)

4. Output the shared secret SS and the ciphertext ct. Send the ciphertext ct to the recipient.

NOTE: The random integer z MUST be generated independently at random for different encryption operations, whether for the same or different recipients.

A.2. Recipient's Operations: RSA-KEM Decapsulate()

Let (n,d) be the recipient's RSA private key; see [RFC8017] for details, but other private key formats are allowed.

Let ct be the ciphertext received from the originator.

Let nLen denote the length in bytes of the modulus n.

The recipient performs the following operations:

1. If the length of the encrypted keying material is less than nLen bytes, output "decryption error", and stop.

2. Convert the ciphertext ct to an integer c, most significant byte first (see NOTE below):

   c = StringToInteger (ct)

   If the integer c is not between 0 and n-1, output "decryption error", and stop.

3. Decrypt the integer c using the recipient's private key (n,d) to recover an integer z (see NOTE below):

   z = c^d mod n

4. Convert the integer z to a byte string Z of length nLen, most significant byte first (see NOTE below):

   Z = IntegerToString (z, nLen)

5. Derive a shared secret SS of length ssLen bytes from the byte string Z using the key-derivation function (see NOTE below):

   SS = KDF (Z, ssLen)

6. Output the shared secret SS.

NOTE: Implementations SHOULD NOT reveal information about the integer z, the string Z, or about the calculation of the exponentiation in Step 2, the conversion in Step 3, or the key derivation in Step 4, whether by timing or other "side channels". The observable behavior of the implementation SHOULD be the same at these steps for all ciphertexts C that are in range. For example, IntegerToString conversion should take the same amount of time regardless of the actual value of the integer z. The integer z, the string Z, and other intermediate results MUST be securely deleted when they are no longer needed.

Appendix B. ASN.1 Syntax

The ASN.1 syntax for identifying the RSA-KEM Algorithm is an extension of the syntax for the "generic hybrid cipher" in ANS X9.44 [ANS-X9.44].

The ASN.1 Module is unchanged from RFC 5990. The id-rsa-kem-spki object identifier is used in a backward compatible manner in certificates [RFC5280] and SMIMECapabilities [RFC8551]. Of course, the use of the id-kem-rsa object identifier in the new KEMRecipientInfo structure [I-D.ietf-lamps-cms-kemri] was not yet defined at the time that RFC 5990 was written.

B.1. Underlying Components

Implementations that conform to this specification MUST support the KDF3 [ANS-X9.44] key-derivation function using SHA-256 [SHS].

KDF2 [ANS-X9.44] and KDF3 are both key-derivation functions based on a hash function. The only difference between KDF2 and KDF3 is the order of the components to be hashed.

KDF2 calculates T as: T = T || Hash (Z || D || otherInfo)

KDF3 calculates T as: T = T || Hash (D || Z || otherInfo)

The object identifier for KDF3 is:

```
id-kdf-kdf3 OBJECT IDENTIFIER ::= { x9-44-components kdf3(2) }
```

The KDF3 parameters identify the underlying hash function. For alignment with the ANS X9.44, the hash function MUST be an ASC X9-approved hash function. While the SHA-1 hash algorithm is included in the ASN.1 definitions, SHA-1 MUST NOT be used. SHA-1 is considered to be obsolete; see [RFC6194]. SHA-1 remains in the ASN.1 module for compatibility with RFC 5990. In addition, other hash functions MAY be used with CMS.

```
kda-kdf3 KEY-DERIVATION ::= {
   IDENTIFIER id-kdf-kdf3
   PARAMS TYPE KDF3-HashFunction ARE required
   -- No S/MIME caps defined -- }

KDF3-HashFunction ::=
   AlgorithmIdentifier { DIGEST-ALGORITHM, {KDF3-HashFunctions} }

KDF3-HashFunctions DIGEST-ALGORITHM ::= { X9-HashFunctions, ... }

X9-HashFunctions DIGEST-ALGORITHM ::= {
   mda-sha1 | mda-sha224 | mda-sha256 | mda-sha384 |
   mda-sha512, ... }
```

Implementations that conform to this specification MUST support the AES Key Wrap [RFC3394] key-encryption algorithm with a 128-bit key. There are three object identifiers for the AES Key Wrap, one for each permitted size of the key-encryption key. There are three object identifiers imported from [RFC5912], and none of these algorithm identifiers have associated parameters:

```
      kwa-aes128-wrap KEY-WRAP ::= {
          IDENTIFIER id-aes128-wrap
          PARAMS ARE absent
          SMIME-CAPS { IDENTIFIED BY id-aes128-wrap } }

      kwa-aes192-wrap KEY-WRAP ::= {
          IDENTIFIER id-aes192-wrap
          PARAMS ARE absent
          SMIME-CAPS { IDENTIFIED BY id-aes192-wrap } }

      kwa-aes256-wrap KEY-WRAP ::= {
          IDENTIFIER id-aes256-wrap
          PARAMS ARE absent
          SMIME-CAPS { IDENTIFIED BY id-aes256-wrap } }
```

B.2. ASN.1 Module

RFC EDITOR: Please replace TBD2 with the value assigned by IANA during the publication of [I-D.ietf-lamps-cms-kemri].

```
<CODE BEGINS>
CMS-RSA-KEM-2023
   { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
     pkcs-9(9) smime(16) modules(0) id-mod-cms-rsa-kem-2023(TBD1) }

   DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS ALL

IMPORTS

  KEM-ALGORITHM
    FROM KEMAlgorithmInformation-2023  -- [I-D.ietf-lamps-cms-kemri]
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
         id-mod-kemAlgorithmInformation-2023(TBD2) }

  AlgorithmIdentifier{}, PUBLIC-KEY, DIGEST-ALGORITHM,
  KEY-DERIVATION, KEY-WRAP, SMIME-CAPS
    FROM AlgorithmInformation-2009  -- [RFC5912]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-algorithmInformation-02(58) }

  kwa-aes128-wrap, kwa-aes192-wrap, kwa-aes256-wrap
    FROM CMSAesRsaesOaep-2009  -- [RFC5911]
      { iso(1) member-body(2) us(840) rsadsi(113549)
        pkcs(1) pkcs-9(9) smime(16) modules(0)
```

```
          id-mod-cms-aes-02(38) }

    kwa-3DESWrap
      FROM CryptographicMessageSyntaxAlgorithms-2009  -- [RFC5911]
        { iso(1) member-body(2) us(840) rsadsi(113549)
          pkcs(1) pkcs-9(9) smime(16) modules(0)
          id-mod-cmsalg-2001-02(37) }

    id-camellia128-wrap, id-camellia192-wrap, id-camellia256-wrap
      FROM CamelliaEncryptionAlgorithmInCMS  -- [RFC3657]
        { iso(1) member-body(2) us(840) rsadsi(113549)
          pkcs(1) pkcs9(9) smime(16) modules(0)
          id-mod-cms-camellia(23) }

    mda-sha1, pk-rsa, RSAPublicKey
      FROM PKIXAlgs-2009  -- [RFC5912]
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-pkix1-algorithms2008-02(56) }

    mda-sha224, mda-sha256, mda-sha384, mda-sha512
      FROM PKIX1-PSS-OAEP-Algorithms-2009  -- [RFC5912]
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-pkix1-rsa-pkalgs-02(54) } ;


  -- Useful types and definitions

  OID ::= OBJECT IDENTIFIER  -- alias

  NullParms ::= NULL

  -- ISO/IEC 18033-2 arc

  is18033-2 OID ::= { iso(1) standard(0) is18033(18033) part2(2) }

  -- NIST algorithm arc

  nistAlgorithm OID ::= { joint-iso-itu-t(2) country(16) us(840)
     organization(1) gov(101) csor(3) nistAlgorithm(4) }

  -- PKCS #1 arc

  pkcs-1 OID ::= { iso(1) member-body(2) us(840) rsadsi(113549)
     pkcs(1) pkcs-1(1) }

  -- X9.44 arc
```

```
   x9-44 OID ::= { iso(1) identified-organization(3) tc68(133)
      country(16) x9(840) x9Standards(9) x9-44(44) }

   x9-44-components OID ::= { x9-44 components(1) }

   -- RSA-KEM Algorithm

   id-rsa-kem OID ::= { iso(1) member-body(2) us(840) rsadsi(113549)
      pkcs(1) pkcs-9(9) smime(16) alg(3) 14 }

   id-rsa-kem-spki OID ::= id-rsa-kem

   GenericHybridParameters ::= SEQUENCE {
      kem  KeyEncapsulationMechanism,
      dem  DataEncapsulationMechanism }

   KeyEncapsulationMechanism ::=
      AlgorithmIdentifier { KEM-ALGORITHM, {KEMAlgorithms} }

   KEMAlgorithms KEM-ALGORITHM ::= { kema-kem-rsa | kema-rsa-kem, ... }

   kema-rsa-kem KEM-ALGORITHM ::= {
      IDENTIFIER id-rsa-kem-spki
      PARAMS TYPE GenericHybridParameters ARE optional
      PUBLIC-KEYS { pk-rsa | pk-rsa-kem }
      UKM ARE optional
      SMIME-CAPS { TYPE GenericHybridParameters
         IDENTIFIED BY id-rsa-kem-spki } }

   kema-kem-rsa KEM-ALGORITHM ::= {
      IDENTIFIER id-kem-rsa
      PARAMS TYPE RsaKemParameters ARE optional
      PUBLIC-KEYS { pk-rsa | pk-rsa-kem }
      UKM ARE optional
      SMIME-CAPS { TYPE GenericHybridParameters
         IDENTIFIED BY id-rsa-kem-spki } }

   id-kem-rsa OID ::= { is18033-2 key-encapsulation-mechanism(2)
      rsa(4) }

   RsaKemParameters ::= SEQUENCE {
      keyDerivationFunction  KeyDerivationFunction,
      keyLength              KeyLength }

   pk-rsa-kem PUBLIC-KEY ::= {
     IDENTIFIER id-rsa-kem-spki
     KEY RSAPublicKey
     PARAMS TYPE GenericHybridParameters ARE preferredAbsent
```

```
     -- Private key format is not specified here --
     CERT-KEY-USAGE {keyEncipherment} }

   KeyDerivationFunction ::=
      AlgorithmIdentifier { KEY-DERIVATION, {KDFAlgorithms} }

   KDFAlgorithms KEY-DERIVATION ::= { kda-kdf2 | kda-kdf3, ... }

   KeyLength ::= INTEGER (1..MAX)

   DataEncapsulationMechanism ::=
      AlgorithmIdentifier { KEY-WRAP, {DEMAlgorithms} }

   DEMAlgorithms KEY-WRAP ::= {
      X9-SymmetricKeyWrappingSchemes |
      Camellia-KeyWrappingSchemes, ... }

   X9-SymmetricKeyWrappingSchemes KEY-WRAP ::= {
      kwa-aes128-wrap | kwa-aes192-wrap | kwa-aes256-wrap |
      kwa-3DESWrap, ... }

   X9-SymmetricKeyWrappingScheme ::=
      AlgorithmIdentifier { KEY-WRAP, {X9-SymmetricKeyWrappingSchemes} }

   Camellia-KeyWrappingSchemes KEY-WRAP ::= {
      kwa-camellia128-wrap | kwa-camellia192-wrap |
      kwa-camellia256-wrap, ... }

   Camellia-KeyWrappingScheme ::=
      AlgorithmIdentifier { KEY-WRAP, {Camellia-KeyWrappingSchemes} }

   kwa-camellia128-wrap KEY-WRAP ::= {
      IDENTIFIER id-camellia128-wrap
      PARAMS ARE absent
      SMIME-CAPS { IDENTIFIED BY id-camellia128-wrap } }

   kwa-camellia192-wrap KEY-WRAP ::= {
      IDENTIFIER id-camellia192-wrap
      PARAMS ARE absent
      SMIME-CAPS { IDENTIFIED BY id-camellia192-wrap } }

   kwa-camellia256-wrap KEY-WRAP ::= {
      IDENTIFIER id-camellia256-wrap
      PARAMS ARE absent
      SMIME-CAPS { IDENTIFIED BY id-camellia256-wrap } }

   -- Key Derivation Functions
```

```
   id-kdf-kdf2 OID ::= { x9-44-components kdf2(1) }

   kda-kdf2 KEY-DERIVATION ::= {
      IDENTIFIER id-kdf-kdf2
      PARAMS TYPE KDF2-HashFunction ARE required
      -- No S/MIME caps defined -- }

   KDF2-HashFunction ::=
      AlgorithmIdentifier { DIGEST-ALGORITHM, {KDF2-HashFunctions} }

   KDF2-HashFunctions DIGEST-ALGORITHM ::= { X9-HashFunctions, ... }

   id-kdf-kdf3 OID ::= { x9-44-components kdf3(2) }

   kda-kdf3 KEY-DERIVATION ::= {
      IDENTIFIER id-kdf-kdf3
      PARAMS TYPE KDF3-HashFunction ARE required
      -- No S/MIME caps defined -- }

   KDF3-HashFunction ::=
      AlgorithmIdentifier { DIGEST-ALGORITHM, {KDF3-HashFunctions} }

   KDF3-HashFunctions DIGEST-ALGORITHM ::= { X9-HashFunctions, ... }

   -- Hash Functions

   X9-HashFunctions DIGEST-ALGORITHM ::= {
      mda-sha1 | mda-sha224 | mda-sha256 | mda-sha384 |
      mda-sha512, ... }

   -- Updates for the SMIME-CAPS Set from RFC 5911

   SMimeCapsSet SMIME-CAPS ::= {
      kema-kem-rsa.&smimeCaps |
      kwa-aes128-wrap |
      kwa-aes192-wrap |
      kwa-aes256-wrap |
      kwa-camellia128-wrap.&smimeCaps |
      kwa-camellia192-wrap.&smimeCaps |
      kwa-camellia256-wrap.&smimeCaps,
      ... }

   END
   <CODE ENDS>
```

Appendix C. SMIMECapabilities Examples

To indicate support for the RSA-KEM algorithm coupled with the KDF3 key-derivation function with SHA-256 and the AES Key Wrap symmetric key-encryption algorithm 128-bit key-encryption key, the SMIMECapabilities will include the following entry:

```
SEQUENCE {
   id-rsa-kem-spki,                              -- RSA-KEM Algorithm
   SEQUENCE {                                 -- GenericHybridParameters
      SEQUENCE {                          -- key encapsulation mechanism
         id-kem-rsa,                                        -- RSA-KEM
         SEQUENCE {                                -- RsaKemParameters
            SEQUENCE {                       -- key derivation function
               id-kdf-kdf3,                                    -- KDF3
               SEQUENCE {                           -- KDF3-HashFunction
                  id-sha256  -- SHA-256; no parameters (preferred)
               },
            16                                    -- KEK length in bytes
            },
      SEQUENCE {                         -- data encapsulation mechanism
         id-aes128-Wrap                -- AES-128 Wrap; no parameters
      }
   }
}
```

This SMIMECapability value has the following DER encoding (in hexadecimal):

```
30 47
  06 0b 2a 86 48 86 f7 0d 01 09 10 03 0e          -- id-rsa-kem-spki
  30 38
     30 29
        06 07 28 81 8c 71 02 02 04                -- id-kem-rsa
        30 1e
           30 19
              06 0a 2b 81 05 10 86 48 09 2c 01 02 -- id-kdf-kdf3
              30 0b
                 06 09 60 86 48 01 65 03 04 02 01 -- id-sha256
                 02 01 10                         -- 16 bytes
      30 0b
         06 09 60 86 48 01 65 03 04 01 05         -- id-aes128-Wrap
```

To indicate support for the RSA-KEM algorithm coupled with the KDF3 key-derivation function with SHA-384 and the AES Key Wrap symmetric key-encryption algorithm 192-bit key-encryption key, the SMIMECapabilities will include the following SMIMECapability value (in hexadecimal):

```
30 47 06 0b 2a 86 48 86 f7 0d 01 09 10 03 0e 30
38 30 29 06 07 28 81 8c 71 02 02 04 30 1e 30 19
06 0a 2b 81 05 10 86 48 09 2c 01 02 30 0b 06 09
60 86 48 01 65 03 04 02 02 02 01 18 30 0b 06 09
60 86 48 01 65 03 04 01 19
```

To indicate support for the RSA-KEM algorithm coupled with the KDF3 key-derivation function with SHA-512 and the AES Key Wrap symmetric key-encryption algorithm 256-bit key-encryption key, the SMIMECapabilities will include the following SMIMECapability value (in hexadecimal):

```
30 47 06 0b 2a 86 48 86 f7 0d 01 09 10 03 0e 30
38 30 29 06 07 28 81 8c 71 02 02 04 30 1e 30 19
06 0a 2b 81 05 10 86 48 09 2c 01 02 30 0b 06 09
60 86 48 01 65 03 04 02 03 02 01 20 30 0b 06 09
60 86 48 01 65 03 04 01 2d
```

Appendix D. RSA-KEM CMS Enveloped-Data Example

This example shows the establishment of an AES-128 content-encryption key using:

* RSA-KEM with a 3072-bit key and KDF3 with SHA-256;

* KEMRecipientInfo key derivation using KDF3 with SHA-256; and

* KEMRecipientInfo key wrap using AES-128-KEYWRAP.

In real-world use, the originator would encrypt the content-encryption key in a manner that would allow decryption with their own private key as well as the recipient's private key. This is omitted in an attempt to simplify the example.

D.1. Originator RSA-KEM Encapsulate() Processing

Alice obtains Bob's public key:

```
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEA3ocW14cxncPJ47fnEjBZ
AyfC2lqapL3ET4jvV6C7gGeVrRQxWPDwl+cFYBBR2ej3j3/0ecDmu+XuVi2+s5JH
Keeza+itfuhsz3yifgeEpeK8T+SusHhn20/NBLhYKbh3kiAcCgQ56dpDrDvDcLqq
vS3jg/VO+OPnZbofoHOOevt8Q/roahJe1PlIyQ4udWB8zZezJ4mLLfbOA9YVaYXx
2AHHZJevo3nmRnlgJXo6mE00E/6qkhjDHKSMdl2WG6mO9TCDZc9qY3cAJDU6Ir0v
SH7qUl8/vN13y4UOFkn8hM4kmZ6bJqbZt5NbjHtY4uQ0VMW3RyESzhrO02mrp39a
uLNnH3EXdXaV1tk75H3qC7zJaeGWMJyQfOE3YfEGRKn8fxubji716D8UecAxAzFy
FL6m1JiOyV5acAiOpxN14qRYZdHnXOM9DqGIGpoeY1UuD4Mo05osOqOUpBJHA9fS
whSZG7VNf+vgNWTLNYSYLI04KiMdulnvU6ds+QPz+KKtAgMBAAE=
-----END PUBLIC KEY-----
```

Bob’s RSA public key has the following key identifier:

```
9eeb67c9b95a74d44d2f16396680e801b5cba49c
```

Alice randomly generates integer z between 0 and n-1:

```
9c126102a5c1c0354672a3c2f19fc9ddea988f815e1da812c7bd4f8eb082bdd1
4f85a7f7c2f1af11d5333e0d6bcb375bf855f208da72ba27e6fb0655f2825aa6
2b93b1f9bbd3491fed58f0380fa0de36430e3a144d569600bd362609be5b9481
0875990b614e406fa6dff500043cbca95968faba61f795096a7fb3687a51078c
4ca2cb663366b0bea0cd9cccac72a25f3f4ed03deb68b4453bba44b943f4367b
67d6cd10c8ace53f545aac50968fc3c6ecc80f3224b64e37038504e2d2c0e2b2
9d45e46c62826d96331360e4c17ea3ef89a9efc5fac99eda830e81450b6534dc
0bdf042b8f3b706649c631fe51fc2445cc8d447203ec2f41f79cdfea16de1ce6
abdfdc1e2ef2e5d5d8a65e645f397240ef5a26f5e4ff715de782e30ecf477293
e89e13171405909a8e04dd31d21d0c57935fc1ceea8e1033e31e1bc8c56da0f3
d79510f3f380ff58e5a61d361f2f18e99fbae5663172e8cd1f21deaddc5bbbea
060d55f1842b93d1a9c888d0bf85d0af9947fe51acf940c7e7577eb79cabecb3
```

Alice encrypts integer z using the Bob’s RSA public key, the result is called ct:

```
c071fc273af8e7bdb152e06bf73310361074154a43abcf3c93c13499d2065344
3eed9ef5d3c0685e4aa76a6854815bb97691ff9f8dac15eea7d74f452bf350a6
46163d68288e978cbf7a73089ee52712f9a4f49e06ace7bbc85ab14d4e336c97
c5728a2654138c7b26e8835c6b0a9fbed26495c4eadf745a2933be283f6a88b1
6695fc06666873cfb6d36718ef3376cefc100c3941f3c494944078325807a559
186b95ccabf3714cfaf79f83bd30537fdd9aed5a4cdcbd8bd0486faed73e9d48
6b3087d6c806546b6e2671575c98461e441f65542bd95de26d0f53a64e7848d7
31d9608d053e8d345546602d86236ffe3704c98ad59144f3089e5e6d527b5497
ba103c79d62e80d0235410b06f71a7d9bd1c38000f910d6312ea2f20a3557535
ad01b3093fb5f7ee507080d0f77d48c9c3b3796f6b7dd3786085fb895123f04c
a1f1c1be22c747a8dface32370fb0d570783e27dbb7e74fca94ee39676fde3d8
a9553d878224736e37e191dab953c7e228c07ad5ca3122421c14debd072a9ab6
```

Alice derives the shared secret (SS) using KDF3 with SHA-256:

3cf82ec41b54ed4d37402bbd8f805a52

D.2. Originator CMS Processing

Alice encodes the CMSORIforKEMOtherInfo structure with the algorithm identifier for AES-128-KEYWRAP and a key length of 16 octets. The DER encoding of CMSORIforKEMOtherInfo produces 18 octets:

3010300b060960864801650304010502 0110

The CMSORIforKEMOtherInfo structure contains:

```
  0  16: SEQUENCE {
  2  11:   SEQUENCE {
  4   9:     OBJECT IDENTIFIER aes128-wrap (2 16 840 1 101 3 4 1 5)
        :     }
 15   1:   INTEGER 16
        :   }
```

Alice derives the key-encryption key from shared secret produced by RSA-KEM Encapsulate() and the CMSORIforKEMOtherInfo structure with KDF3 and SHA-256, the KEK is:

e6dc9d62ff2b469bef604c617b018718

Alice randomly generates a 128-bit content-encryption key:

77f2a84640304be7bd42670a84a1258b

Alice uses AES-128-KEYWRAP to encrypt the 128-bit content-encryption key with the derived key-encryption key:

28782e5d3d794a7616b863fbcfc719b78f12de08cf286e09

Alice encrypts the padded content using AES-128-CBC with the content-encryption key. The 16-octet IV used is:

480ccafebabefacedbaddecaf8887781

The padded content plaintext is:

48656c6c6f2c20776f726c6421030303

The resulting ciphertext is:

c6ca65db7bdd76b0f37e2fab6264b66d

Alice encodes the EnvelopedData (using KEMRecipientInfo) and ContentInfo, and then sends the result to Bob. The Base64-encoded result is:

```
MIICXAYJKoZIhvcNAQcDoIICTTCCAkkCAQMxggIEpIICAAYLKoZIhvcNAQkQDQMw
ggHvAgEAgBSe62fJuVp01E0vFjlmgOgBtcuknDAJBgcogYxxAgIEBIIBgMBx/Cc6
+Oe9sVLga/czEDYQdBVKQ6vPPJPBNJnSBlNEPu2e9dPAaF5Kp2poVIFbuXaR/5+N
rBXup9dPRSvzUKZGFj1oKI6XjL96cwie5ScS+aT0ngas57vIWrFNTjNsl8VyiiZU
E4x7JuiDXGsKn77SZJXE6t90Wikzvig/aoixZpX8BmZoc8+202cY7zN2zvwQDDlB
88SUlEB4MlgHpVkYa5XMq/NxTPr3n4O9MFN/3ZrtWkzcvYvQSG+u1z6dSGswh9bI
BlRrbiZxV1yYRh5EH2VUK9ld4m0PU6ZOeEjXMdlgjQU+jTRVRmAthiNv/jcEyYrV
kUTzCJ5ebVJ7VJe6EDx51i6A0CNUELBvcafZvRw4AA+RDWMS6i8go1V1Na0Bswk/
tffuUHCA0Pd9SMnDs3lva33TeGCF+4lRI/BMofHBviLHR6jfrOMjcPsNVweD4n27
fnT8qU7jlnb949ipVT2HgiRzbjfhkdq5U8fiKMB61coxIkIcFN69ByqatjAbBgor
gQUQhkgJLAECMA0GCWCGSAFlAwQCAQUAAgEQMAsGCWCGSAFlAwQBBQQYKHguXT15
SnYWuGP7z8cZt48S3gjPKG4JMDwGCSqGSIb3DQEHATAdBglghkgBZQMEAQIEEEgM
yv66vvrO263eyviId4GAEMbKZdt73Xaw834vq2Jktm0=
```

This result decodes to:

```
  0 604: SEQUENCE {
  4   9:  OBJECT IDENTIFIER envelopedData (1 2 840 113549 1 7 3)
 15 589:  [0] {
 19 585:   SEQUENCE {
 23   1:    INTEGER 3
 26 516:    SET {
 30 512:     [4] {
 34  11:      OBJECT IDENTIFIER
       :       KEMRecipientInfo (1 2 840 113549 1 9 16 13 3)
 47 495:      SEQUENCE {
 51   1:       INTEGER 0
 54  20:       [0]
       :       9E EB 67 C9 B9 5A 74 D4 4D 2F 16 39 66 80 E8 01
       :       B5 CB A4 9C
 76   9:       SEQUENCE {
 78   7:        OBJECT IDENTIFIER kemRSA (1 0 18033 2 2 4)
       :         }
 87 384:       OCTET STRING
       :       C0 71 FC 27 3A F8 E7 BD B1 52 E0 6B F7 33 10 36
       :       10 74 15 4A 43 AB CF 3C 93 C1 34 99 D2 06 53 44
       :       3E ED 9E F5 D3 C0 68 5E 4A A7 6A 68 54 81 5B B9
       :       76 91 FF 9F 8D AC 15 EE A7 D7 4F 45 2B F3 50 A6
       :       46 16 3D 68 28 8E 97 8C BF 7A 73 08 9E E5 27 12
       :       F9 A4 F4 9E 06 AC E7 BB C8 5A B1 4D 4E 33 6C 97
       :       C5 72 8A 26 54 13 8C 7B 26 E8 83 5C 6B 0A 9F BE
       :       D2 64 95 C4 EA DF 74 5A 29 33 BE 28 3F 6A 88 B1
       :       66 95 FC 06 66 68 73 CF B6 D3 67 18 EF 33 76 CE
       :       FC 10 0C 39 41 F3 C4 94 94 40 78 32 58 07 A5 59
```

```
          :       18 6B 95 CC AB F3 71 4C FA F7 9F 83 BD 30 53 7F
          :       DD 9A ED 5A 4C DC BD 8B D0 48 6F AE D7 3E 9D 48
          :       6B 30 87 D6 C8 06 54 6B 6E 26 71 57 5C 98 46 1E
          :       44 1F 65 54 2B D9 5D E2 6D 0F 53 A6 4E 78 48 D7
          :       31 D9 60 8D 05 3E 8D 34 55 46 60 2D 86 23 6F FE
          :       37 04 C9 8A D5 91 44 F3 08 9E 5E 6D 52 7B 54 97
          :       BA 10 3C 79 D6 2E 80 D0 23 54 10 B0 6F 71 A7 D9
          :       BD 1C 38 00 0F 91 0D 63 12 EA 2F 20 A3 55 75 35
          :       AD 01 B3 09 3F B5 F7 EE 50 70 80 D0 F7 7D 48 C9
          :       C3 B3 79 6F 6B 7D D3 78 60 85 FB 89 51 23 F0 4C
          :       A1 F1 C1 BE 22 C7 47 A8 DF AC E3 23 70 FB 0D 57
          :       07 83 E2 7D BB 7E 74 FC A9 4E E3 96 76 FD E3 D8
          :       A9 55 3D 87 82 24 73 6E 37 E1 91 DA B9 53 C7 E2
          :       28 C0 7A D5 CA 31 22 42 1C 14 DE BD 07 2A 9A B6
      475  27:       SEQUENCE {
      477  10:        OBJECT IDENTIFIER
          :          kdf3 (1 3 133 16 840 9 44 1 2)
      489  13:        SEQUENCE {
      491   9:         OBJECT IDENTIFIER
          :           sha-256 (2 16 840 1 101 3 4 2 1)
      502   0:         NULL
          :          }
          :         }
      504   1:       INTEGER 16
      507  11:       SEQUENCE {
      509   9:        OBJECT IDENTIFIER
          :          aes128-wrap (2 16 840 1 101 3 4 1 5)
          :         }
      520  24:       OCTET STRING
          :       28 78 2E 5D 3D 79 4A 76 16 B8 63 FB CF C7 19 B7
          :       8F 12 DE 08 CF 28 6E 09
          :        }
          :       }
          :      }
      546  60:    SEQUENCE {
      548   9:     OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
      559  29:     SEQUENCE {
      561   9:      OBJECT IDENTIFIER
          :       aes128-CBC (2 16 840 1 101 3 4 1 2)
      572  16:      OCTET STRING
          :       48 0C CA FE BA BE FA CE DB AD DE CA F8 88 77 81
          :       }
      590  16:     [0] C6 CA 65 DB 7B DD 76 B0 F3 7E 2F AB 62 64 B6 6D
          :      }
          :     }
          :    }
          :   }
```

D.3. Recipient RSA-KEM Decapsulate() Processing

Bob's private key:

```
-----BEGIN PRIVATE KEY-----
MIIG5AIBAAKCAYEA3ocW14cxncPJ47fnEjBZAyfC2lqapL3ET4jvV6C7gGeVrRQx
WPDwl+cFYBBR2ej3j3/0ecDmu+XuVi2+s5JHKeeza+itfuhsz3yifgeEpeK8T+Su
sHhn20/NBLhYKbh3kiAcCgQ56dpDrDvDcLqqvS3jg/VO+OPnZbofoHOOevt8Q/ro
ahJe1PlIyQ4udWB8zZezJ4mLLfbOA9YVaYXx2AHHZJevo3nmRnlgJXo6mE00E/6q
khjDHKSMdl2WG6mO9TCDZc9qY3cAJDU6Ir0vSH7qUl8/vN13y4UOFkn8hM4kmZ6b
JqbZt5NbjHtY4uQ0VMW3RyESzhrO02mrp39auLNnH3EXdXaV1tk75H3qC7zJaeGW
MJyQfOE3YfEGRKn8fxubji716D8UecAxAzFyFL6m1JiOyV5acAiOpxN14qRYZdHn
XOM9DqGIGpoeY1UuD4Mo05osOqOUpBJHA9fSwhSZG7VNf+vgNWTLNYSYLI04KiMd
ulnvU6ds+QPz+KKtAgMBAAECggGATFfkSkUjjJCjLvDk4aScpSx6+Rakf2hrdS3x
jwqhyUfAXgTTeUQQBs1HVtHCgxQd+qlXYn3/qu8TeZVwG4NPztyi/Z5yB1wOGJEV
3k8N/ytul6pJFFn6p48VM01bUdTrkMJbXERe6g/rr6dBQeeItCaOK7N5SIJH3Oqh
9xYuB5tH4rquCdYLmt17Tx8CaVqU9qPY3vOdQEOwIjjMV8uQUR8rHSO9KkSj8AGs
Lq9kcuPpvgJc2oqMRcNePS2WVh8xPFktRLLRazgLP8STHAtjT6SlJ2UzkUqfDHGK
q/BoXxBDu6L1VDwdnIS5HXtL54ElcXWsoOyKF8/ilmhRUIUWRZFmlS1ok8IC5IgX
UdL9rJVZFTRLyAwmcCEvRM1asbBrhyEyshSOuN5nHJi2WVJ+wSHijeKl1qeLlpMk
HrdIYBq4Nz7/zXmiQphpAy+yQeanhP8O4O6C8e7RwKdpxe44su4Z8fEgA5yQx0u7
8yR1EhGKydX5bhBLR5Cm1VM7rT2BAoHBAP/+e5gZLNf/ECtEBZjeiJ0VshszOoUq
haUQPA+9Bx9pytsoKm5oQhB7QDaxAvrn8/FUW2aAkaXsaj9F+/q30AYSQtExai9J
fdKKook3oimN8/yNRsKmhfjGOj8hd4+GjX0qoMSBCEVdT+bAjjry8wgQrqReuZnu
oXU85dmb3jvv0uIczIKvTIeyjXE5afjQIJLmZFXsBm09BG87Ia5EFUKly96BOMJh
/QWEzuYYXDqOFfzQtkAefXNFW21Kz4Hw2QKBwQDeiGh4lxCGTjECvG7fauMGlu+q
DSdYyMHif6t6mx57eS16EjvOrlXKItYhIyzW8Kw0rf/CSB2j8ig1GkMLTOgrGIJ1
0322o50FOr5oOmZPueeR4pOyAP0fgQ8DD1L3JBpY68/8MhYbsizVrR+Ar4jM0f96
W2bF5Xj3h+fQTDMkx6VrCCQ6miRmBUzH+ZPs5n/lYOzAYrqiKOanaiHy4mjRvlsy
mjZ6z5CG8sISqcLQ/k3Qli5pOY/v0rdBjgwAW/UCgcEAqGVYGjKdXCzuDvf9EpV4
mpTWB6yIV2ckaPOn/tZi5BgsmEPwvZYZt0vMbu28Px7sSpkqUuBKbzJ4pcy8uC3I
SuYiTAhMiHS4rxIBX3BYXSuDD2RD4vG1+XM0h6jVRHXHh0nOXdVfgnmigPGz3jVJ
B8oph/jD8O2YCk4YCTDOXPEi8Rjusxzro+whvRR+kG0gsGGcKSVNCPj1fNISEte4
gJId7O1mUAAzeDjn/VaS/PXQovEMolssPPKn9NocbKbpAoHBAJnFHJunl22W/lrr
ppmPnIzjI30YVcYOA5vlqLKyGaAsnfYqP1WUNgfVhq2jRsrHx9cnHQI9Hu442PvI
x+c5H30YFJ4ipE3eRRRmAUi4ghY5WgD+1hw8fqyUW7E7l5LbSbGEUVXtrkU5G64T
UR91LEyMF8OPATdiV/KD4PWYkgaqRm3tVEuCVACDTQkqNsOOi3YPQcm270w6gxfQ
SOEy/kdhCFexJFA8uZvmh6Cp2crczxyBilR/yCxqKOONqlFdOQKBwFbJk5eHPjJz
AYueKMQESPGYCrwIqxgZGCxaqeVArHvKsEDx5whI6JWoFYVkFA8F0MyhukoEb/2x
2qB5T88Dg3EbqjTiLg3qxrWJ2OxtUo8pBP2I2wbl2NOwzcbrlYhzEZ8bJyxZu5i1
sYILC8PJ4Qzw6jS4Qpm4y1WHz8e/ElW6VyfmljZYA7f9WMntdfeQVqCVzNTvKn6f
hg6GSpJTzp4LV3ougi9nQuWXZF2wInsXkLYpsiMbL6Fz34RwohJtYA==
-----END PRIVATE KEY-----
```

Bob checks that the length of the ciphertext is less than nLen bytes.

Bob checks that the ciphertext is greater than zero and is less than his RSA modulus.

Bob decrypts the ciphertext with his RSA private key to obtain the integer z:

```
9c126102a5c1c0354672a3c2f19fc9ddea988f815e1da812c7bd4f8eb082bdd1
4f85a7f7c2f1af11d5333e0d6bcb375bf855f208da72ba27e6fb0655f2825aa6
2b93b1f9bbd3491fed58f0380fa0de36430e3a144d569600bd362609be5b9481
0875990b614e406fa6dff500043cbca95968faba61f795096a7fb3687a51078c
4ca2cb663366b0bea0cd9cccac72a25f3f4ed03deb68b4453bba44b943f4367b
67d6cd10c8ace53f545aac50968fc3c6ecc80f3224b64e37038504e2d2c0e2b2
9d45e46c62826d96331360e4c17ea3ef89a9efc5fac99eda830e81450b6534dc
0bdf042b8f3b706649c631fe51fc2445cc8d447203ec2f41f79cdfea16de1ce6
abdfdc1e2ef2e5d5d8a65e645f397240ef5a26f5e4ff715de782e30ecf477293
e89e13171405909a8e04dd31d21d0c57935fc1ceea8e1033e31e1bc8c56da0f3
d79510f3f380ff58e5a61d361f2f18e99fbae5663172e8cd1f21deaddc5bbbea
060d55f1842b93d1a9c888d0bf85d0af9947fe51acf940c7e7577eb79cabecb3
```

Bob checks that the integer z is greater than zero and is less than his RSA modulus.

Bob derives the shared secret (SS) using KDF3 with SHA-256:

```
3cf82ec41b54ed4d37402bbd8f805a52
```

D.4. Recipient CMS Processing

Bob encodes the CMSORIforKEMOtherInfo structure with the algorithm identifier for AES-128-KEYWRAP and a key length of 16 octets. The DER encoding of CMSORIforKEMOtherInfo is not repeated here.

Bob derives the key-encryption key from shared secret and the CMSORIforKEMOtherInfo structure with KDF3 and SHA-256, the KEK is:

```
e6dc9d62ff2b469bef604c617b018718
```

Bob uses AES-KEY-WRAP to decrypt the content-encryption key with the key-encryption key; the content-encryption key is:

```
77f2a84640304be7bd42670a84a1258b
```

Bob decrypts the content using AES-128-CBC with the content-encryption key. The 16-octet IV used is:

```
480ccafebabefacedbaddecaf8887781
```

The received ciphertext content is:

```
c6ca65db7bdd76b0f37e2fab6264b66d
```

The resulting padded plaintext content is:

48656c6c6f2c20776f726c6421030303

After stripping the AES-CBC padding, the plaintext content is:

Hello, world!

Acknowledgements

We thank James Randall, Burt Kaliski, and John Brainard as the original authors of [RFC5990]; this document is based on their work.

We thank the members of the ASC X9F1 working group for their contributions to drafts of ANS X9.44, which led to [RFC5990].

We thank Blake Ramsdell, Jim Schaad, Magnus Nystrom, Bob Griffin, and John Linn for helping bring [RFC5990] to fruition.

We thank Burt Kaliski, Alex Railean, Joe Mandel, Mike Ounsworth, Peter Campbell, and Daniel Van Geest for careful review and thoughtful comments that greatly improved this document.

Authors' Addresses

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA, 20170
United States of America
Email: housley@vigilsec.com

Sean Turner
sn3rd
Email: sean@sn3rd.com

MPLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 20 October 2024

IJ. Wijnands
Individual
M. Mishra (Editor)
K. Raza
Cisco Systems, Inc.
Z. Zhang
Juniper Networks
A. Gulko
Edward Jones wealth management

18 April 2024

mLDP Extensions for Multi-Topology Routing
draft-ietf-mpls-mldp-multi-topology-05

Abstract

Multi-Topology Routing (MTR) is a technology to enable service differentiation within an IP network. Flexible Algorithm (FA) is another mechanism of creating a sub-topology within a topology using defined topology constraints and computation algorithm. In order to deploy mLDP (Multipoint label distribution protocol) in a network that supports MTR and/or FA, mLDP is required to become topology and FA aware. This document specifies extensions to mLDP to support MTR with FA such that when building a Multipoint LSPs(Label Switched Paths) it can follow a particular topology and algorithm.

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 20 October 2024.

Table of Contents

1. Glossary

MT - Multi-Topology

MT-ID - Multi-Topology Identifier

MTR - Multi-Topology Routing

IGP - Interior Gateway Protocol

MP - Multipoint (P2MP or MP2MP)

LDP - Label Distribution Protocol

mLDP - Multipoint LDP

P2MP - Point-to-Multipoint

MP2MP - Multipoint-to-Multipoint

FEC - Forwarding Equivalence Class

LSP - Label Switched Path

FA - Flexible Algorithm

IPA - IGP Algorithm

2. Introduction

Multi-Topology Routing (MTR) is a technology to enable service differentiation within an IP network. IGP protocols (OSPF and IS-IS) and LDP have already been extended to support MTR. To support MTR, an IGP maintains independent IP topologies, termed as "Multi-Topologies" (MT), and computes/installs routes per topology. OSPF extensions [RFC4915] and ISIS extensions [RFC5120] specify the MT extensions under respective IGPs. To support IGP MT, similar LDP extensions [RFC7307] have been specified to make LDP MT-aware and be able to setup unicast Label Switched Paths (LSPs) along IGP MT routing paths.

A more light weight mechanism to define constraint-based topologies is the Flexible Algorithm (FA) [RFC9350]. FA can be seen as creating a sub-topology within a topology using defined topology constraints and computation algorithm. This can be done within a MTR topology or the default Topology. An instance of such a sub-topology is identified by a 1 octet value as documented in [RFC9350]). Flexible Algorithm is a mechanism to create a sub-topology, but in the future different algorithms might be defined on how to achieve that. For that reason, in the remainder of this document, we'll refer to this as the IGP Algorithm (IPA).

Multipoint LDP (mLDP) refers to extensions in LDP to setup multi-point LSPs (point-to-multipoint (P2MP) or multipoint-to-multipoint (MP2MP)), by means of set of extensions and procedures defined in [RFC6388]. In order to deploy mLDP in a network that supports MTR and FA, mLDP is required to become topology and algorithm aware. This document specifies extensions to mLDP to support MTR/IPA such

that when building a Multi-Point LSPs it can follow a particular topology and alogirthm. This means that the identifier for the particular topology to be used by mLDP have to become a 2-tuple (MTR Topology Id, IGP Algorithm).

3. Specification of Requirements

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.

4. MT Scoped mLDP FECs

As defined in [RFC7307], MPLS Multi-Topology Identifier (MT-ID) is an identifier that is used to associate an LSP with a certain MTR topology. In the context of MP LSPs, this identifier is part of the mLDP FEC encoding so that LDP peers are able to setup an MP LSP via their own defined MTR policy. In order to avoid conflicting MTR policies for the same mLDP FEC, the MT-ID needs to be a part of the FEC, so that different MT-ID values will result in unique MP-LSP FEC elements.

The same applies to the IPA. The IPA needs to be encoded as part of the mLDP FEC to create unique MP-LSPs and at the same time is used to signal to mLDP (hop-by-hop) which Algorithm needs to be used to create the MP-LSP.

Since the MT-ID and IPA are part of the FEC, they apply to all the LDP messages that potentially include an mLDP FEC element.

4.1. MP FEC Extensions for MT

Following subsections propose the extensions to bind an mLDP FEC to a topology. The mLDP MT extensions reuse some of the extensions specified in [RFC7307].

4.1.1. MP FEC Element

Base mLDP specification [RFC6388] defines MP FEC Element as follows:

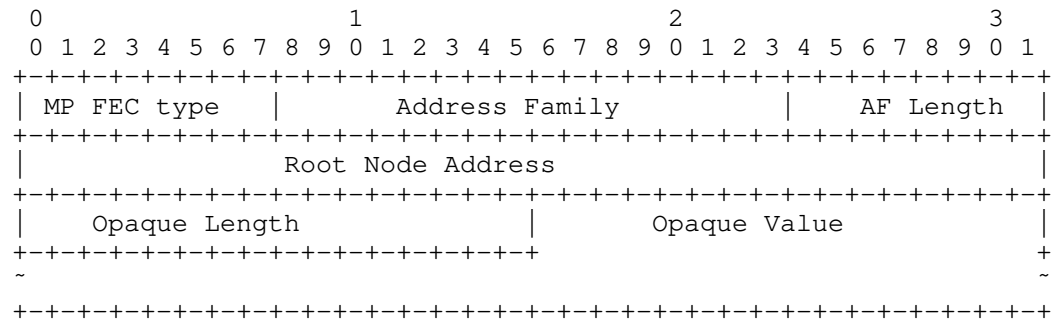


Figure 1: MP FEC Element Format [RFC6388]

Where "Root Node Address" encoding is as defined for given "Address Family", and whose length (in octets) is specified by the "AF Length" field.

To extend MP FEC elements for MT, the {MT-ID, IPA} is a tuple that is relevant in the context of the root address of the MP LSP. The {MT-ID, IPA} tuple determines in which (sub)-topology the root address needs to be resolved. Since the {MT-ID, IPA} tuple should be considered part of the mLDP FEC, the most natural place to encode this tuple is as part of the root address. While encoding it, we also propose to use "MT IP" Address Families as described in following sub section.

4.1.2. MT IP Address Families

[RFC7307] has specified new address families, named "MT IP" and "MT IPv6", to allow specification of an IP prefix within a topology scope. In addition to using this address family for mLDP, we also use 8 bits of the 16 bits Reserved field to encode the IGP Algorithm (IPA) Registry. The resulting format of the data associated with these new Address Families is as follows:

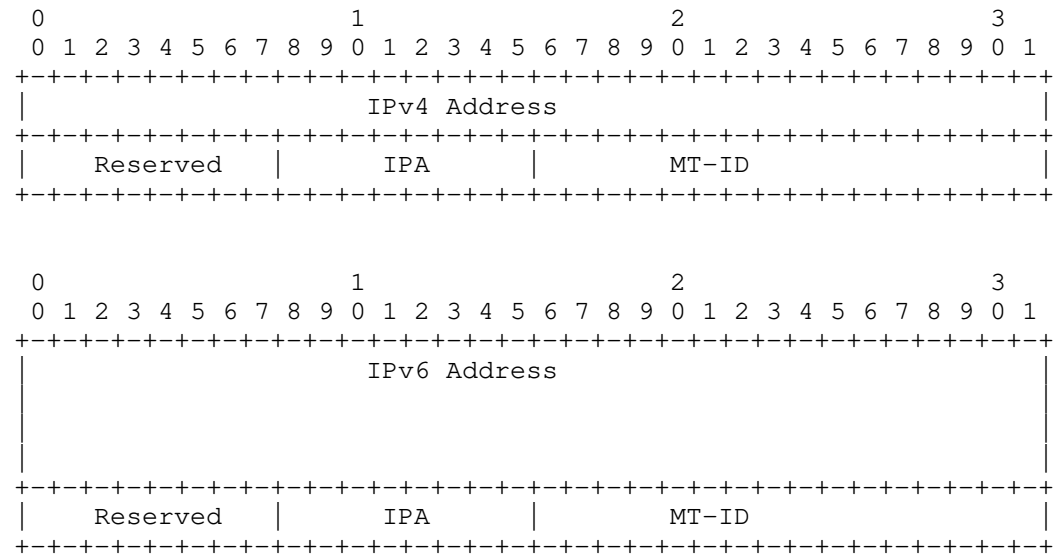


Figure 2: Modified MT IP Address Families Data Format

Where:

IPv4/IPv6 Address: An IP address corresponding to "MT IP" and "MT IPv6" address families respectively.

IPA: The IGP Algorithm, values are from the IGP Algorithm registry.

Reserved: This 8-bit field MUST be zero on transmission and ignored on receipt.

4.1.3. MT MP FEC Element

By using extended MT IP Address Family, the resultant MT MP FEC element is to be encoded as follows:

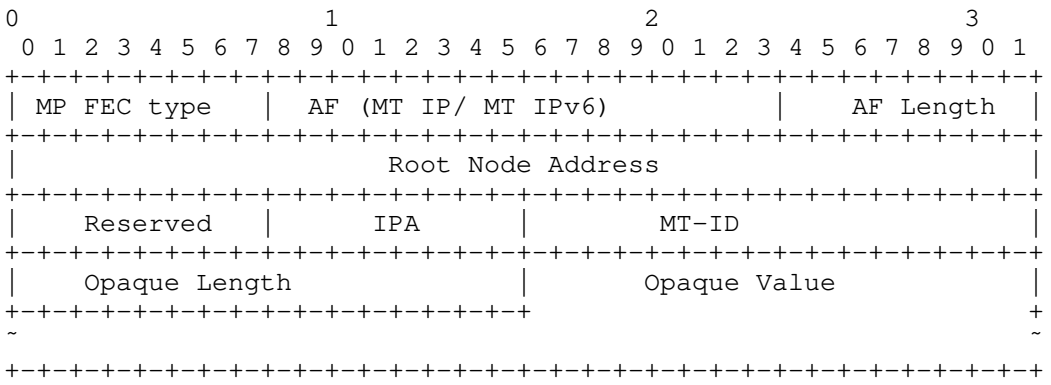


Figure 3: IP MT-Scoped MP FEC Element Format

In the context of this document, the applicable LDP FECs for MT mLDP include:

* MP FEC Elements:

  - P2MP (type 0x6)

  - MP2MP-up (type 0x7)

  - MP2MP-down (type 0x8)

* Typed Wildcard FEC Element (type 0x5)

In case of "Typed Wildcard FEC Element", the sub FEC Element type MUST be one of the MP FECs listed above.

This specification allows the use of Topology-scoped mLDP FECs in LDP label and notification messages, as applicable.

[RFC6514] defines the PMSI tunnel attribute for MVPN. When the Tunnel Type is set to mLDP P2MP LSP, the Tunnel Identifier is a P2MP FEC Element. When the Tunnel Type is set to mLDP Multipoint-to-Multipoint (MP2MP) LSP, the Tunnel Identifier is an MP2MP FEC Element. For deploying mLDP in a network that supports MTR and FA, New MT MP FEC element SHOULD be used as the Tunnel Identifier.

4.2. Topology IDs

This document assumes the same definitions and procedures associated with MPLS MT-ID as defined in [RFC7307] specification.

5. MT Multipoint Capability

"MT Multipoint Capability" is a new LDP capability, defined in accordance with LDP Capability definition guidelines [RFC5561], that is to be advertised to its peers by an mLDP speaker to announce its capability to support MTR and the procedures specified in this document. This capability MAY be sent either in an Initialization message at the session establishment time, or in a Capability message dynamically during the lifetime of a session (only if "Dynamic Announcement" capability [RFC5561] has been successfully negotiated with the peer).

The format of this capability is as follows:

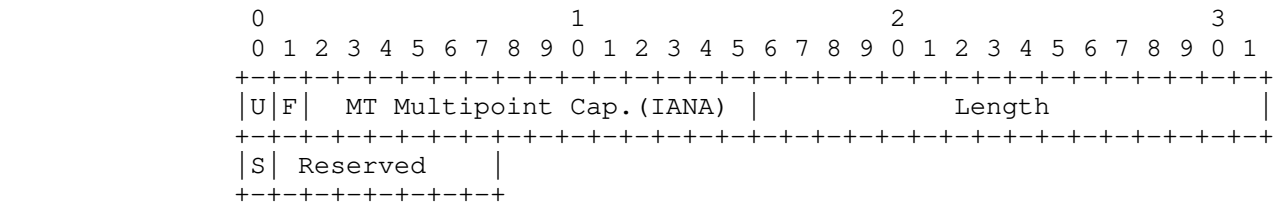


Figure 4: MT Multipoint Capability TLV Format

Where:

U- and F-bits: MUST be 1 and 0, respectively, as per Section 3 of LDP Capabilities [RFC5561].

MT Multipoint Capaility: TLV type (IANA assigned).

Length: The length (in octets) of TLV. The value of this field MUST be 1 as there is no Capability-specific data [RFC5561] that follows in the TLV.

S-bit: Set to 1 to announce and 0 to withdraw the capability (as per [RFC5561].

An mLDP speaker that has successfully advertised and negotiated "MT Multipoint" capability MUST support the following:

1. Topology-scoped mLDP FECs in LDP messages (Section 4.1)

2. Topology-scoped mLDP forwarding setup (Section 7)

6. MT Applicability on FEC-based features

6.1. Typed Wildcard MP FEC Elements

[RFC5918] extends base LDP and defines Typed Wildcard FEC Element framework. Typed Wildcard FEC element can be used in any LDP message to specify a wildcard operation for a given type of FEC.

The MT extensions proposed in document do not require any extension in procedures for Typed Wildcard FEC Element support [RFC5918], and these procedures apply as-is to Multipoint MT FEC wildcarding. Like Typed Wildcard MT Prefix FEC Element, as defined in [RFC7307], the MT extensions allow use of "MT IP" or "MT IPv6" in the Address Family field of the Typed Wildcard MP FEC element in order to use wildcard operations for MP FECs in the context of a given (sub)-topology as identified by the MT-ID and IPA field.

This document proposes following format and encoding for a Typed Wildcard MP FEC element:

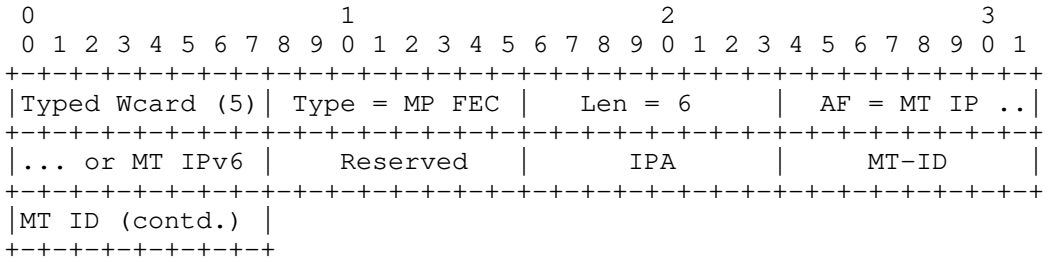


Figure 5: Typed Wildcard MT MP FEC Element

Where:

Type: One of MP FEC Element type (P2MP, MP2MPup, MP2MP-down).

MT ID: MPLS MT ID

IPA: The IGP Algorithm, values are from the IGP Algorithm registry.

The proposed format allows an LSR to perform wildcard MP FEC operations under the scope of a (sub-)topology.

6.2. End-of-LIB

[RFC5919] specifies extensions and procedures that allows an LDP speaker to signal its End-of-LIB (i.e. convergence) for a given FEC type towards a peer. MT extensions for MP FEC do not require any change in these procedures and they apply as-is to MT MP FEC elements. This means that an MT mLDP speaker MAY signal its convergence per (sub-)topology using MT Typed Wildcard MP FEC element.

7. Topology-Scoped Signaling and Forwarding

Since the {MT-ID, IPA} tuple is part of an mLDP FEC, there is no need to support the concept of multiple (sub-)topology forwarding tables in mLDP. Each MP LSP will be unique due to the tuple being part of the FEC. There is also no need to have specific label forwarding tables per topology, and each MP LSP will have its own unique local label in the table. However, In order to implement MTR in an mLDP network, the selection procedures for upstream LSR and downstream forwarding interface need to be changed.

7.1. Upstream LSR selection

The procedures as described in RFC-6388 section-2.4.1.1 depend on the best path to reach the root. When the {MT-ID, IPA} tuple is signaled as part of the FEC, this tuple is used to select the (sub-)topology that must be used to find the best path to the root address. Using the next-hop from this best path, a LDP peer is selected following the procedures as defined in [RFC6388].

7.2. Downstream forwarding interface selection

The procedures as described in RFC-6388 section-2.4.1.2 describe how a downstream forwarding interface is selected. In these procedures, any interface leading to the downstream LDP neighbor can be considered as candidate forwarding interface. When the {MT-ID, IPA} tuple is part of the FEC, this is no longer true. An interface must only be selected if it is part of the same (sub-)topology that was signaled in the mLDP FEC element. Besides this restriction, the other procedures in [RFC6388] apply.

8. LSP Ping Extensions

[RFC6425] defines procedures to detect data plane failures in Multipoint MPLS LSPs. Section 3.1.2 of [RFC6425] defines new Sub-Types and Sub-TLVs for Multipoint LDP FECs to be sent in "Target FEC Stack" TLV of an MPLS echo request message [RFC8029].

To support LSP ping for MT Multipoint LSPs, this document uses existing sub-types "P2MP LDP FEC Stack" and "MP2MP LDP FEC Stack" defined in [RFC6425]. The proposed extension is to specify "MT IP" or "MT IPv6" in the "Address Family" field, set the "Address Length" field to 8 (for MT IP) or 20 (for MT IPv6), and encode the sub-TLV with additional {MT-ID, IPA} information as an extension to the "Root LSR Address" field. The resultant format of sub-tlv is as follows:

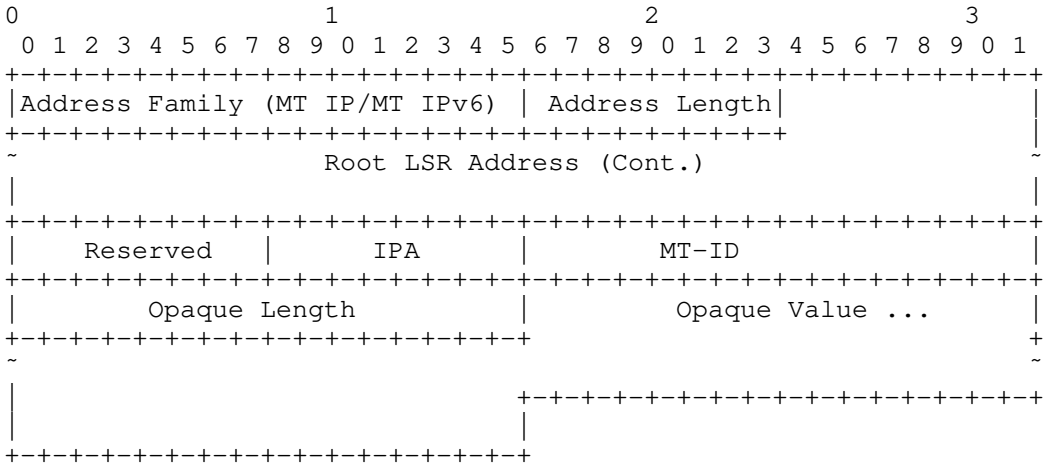


Figure 6: Multipoint LDP FEC Stack Sub-TLV Format for MT

The rules and procedures of using this new sub-TLV in an MPLS echo request message are same as defined for P2MP/MP2MP LDP FEC Stack Sub-TLV in [RFC6425] with only difference being that Root LSR address is now (sub-)topology scoped.

9. Implementation Status

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

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942] . The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942] , "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. Cisco Systems

The feature has been implemented on IOS-XR.

* Organization: Cisco Systems
* Implementation: Cisco systems IOS-XR has an implementation
* Description: The implementation has been done.
* Maturity Level: Product
* Contact: mankamis@cisco.com

10. Security Considerations

This extension to mLDP does not introduce any new security considerations beyond that already apply to the base LDP specification [RFC5036], base mLDP specification [RFC6388], and MPLS security framework [RFC5920].

11. IANA Considerations

This document defines a new LDP capability parameter TLV. IANA is requested to assign the lowest available value after 0x0500 from "TLV Type Name Space" in the "Label Distribution Protocol (LDP) Parameters" registry within "Label Distribution Protocol (LDP) Name Spaces" as the new code point for the LDP TLV code point.

Value	Description	Reference	Notes/Registration Date
TBA	MT Multipoint Capability	This document	

Figure 7: IANA Code Point

12. Contributor

Anuj Budhiraja Cisco systems

13. Acknowledgments

The authors would like to acknowledge Eric Rosen for his input on this specification.

14. References

14.1. 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>.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, DOI 10.17487/RFC4915, June 2007, <https://www.rfc-editor.org/info/rfc4915>.

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <https://www.rfc-editor.org/info/rfc5120>.

[RFC6388] Wijnands, IJ., Ed., Minei, I., Ed., Kompella, K., and B. Thomas, "Label Distribution Protocol Extensions for Point-to-Multipoint and Multipoint-to-Multipoint Label Switched Paths", RFC 6388, DOI 10.17487/RFC6388, November 2011, <https://www.rfc-editor.org/info/rfc6388>.

[RFC6425] Saxena, S., Ed., Swallow, G., Ali, Z., Farrel, A., Yasukawa, S., and T. Nadeau, "Detecting Data-Plane Failures in Point-to-Multipoint MPLS - Extensions to LSP Ping", RFC 6425, DOI 10.17487/RFC6425, November 2011, <https://www.rfc-editor.org/info/rfc6425>.

[RFC6514] Aggarwal, R., Rosen, E., Morin, T., and Y. Rekhter, "BGP Encodings and Procedures for Multicast in MPLS/BGP IP VPNs", RFC 6514, DOI 10.17487/RFC6514, February 2012, <https://www.rfc-editor.org/info/rfc6514>.

[RFC7307] Zhao, Q., Raza, K., Zhou, C., Fang, L., Li, L., and D. King, "LDP Extensions for Multi-Topology", RFC 7307, DOI 10.17487/RFC7307, July 2014, <https://www.rfc-editor.org/info/rfc7307>.

[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>.

[RFC8029] Kompella, K., Swallow, G., Pignataro, C., Ed., Kumar, N., Aldrin, S., and M. Chen, "Detecting Multiprotocol Label Switched (MPLS) Data-Plane Failures", RFC 8029, DOI 10.17487/RFC8029, March 2017, <https://www.rfc-editor.org/info/rfc8029>.

[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>.

[RFC9350] Psenak, P., Ed., Hegde, S., Filsfils, C., Talaulikar, K., and A. Gulko, "IGP Flexible Algorithm", RFC 9350, DOI 10.17487/RFC9350, February 2023, <https://www.rfc-editor.org/info/rfc9350>.

14.2. Informative References

[RFC5036] Andersson, L., Ed., Minei, I., Ed., and B. Thomas, Ed., "LDP Specification", RFC 5036, DOI 10.17487/RFC5036, October 2007, <https://www.rfc-editor.org/info/rfc5036>.

[RFC5561] Thomas, B., Raza, K., Aggarwal, S., Aggarwal, R., and JL. Le Roux, "LDP Capabilities", RFC 5561, DOI 10.17487/RFC5561, July 2009, <https://www.rfc-editor.org/info/rfc5561>.

[RFC5918] Asati, R., Minei, I., and B. Thomas, "Label Distribution Protocol (LDP) 'Typed Wildcard' Forward Equivalence Class (FEC)", RFC 5918, DOI 10.17487/RFC5918, August 2010, <https://www.rfc-editor.org/info/rfc5918>.

[RFC5919] Asati, R., Mohapatra, P., Chen, E., and B. Thomas, "Signaling LDP Label Advertisement Completion", RFC 5919, DOI 10.17487/RFC5919, August 2010, <https://www.rfc-editor.org/info/rfc5919>.

[RFC5920] Fang, L., Ed., "Security Framework for MPLS and GMPLS Networks", RFC 5920, DOI 10.17487/RFC5920, July 2010, <https://www.rfc-editor.org/info/rfc5920>.

Authors' Addresses

IJsbrand Wijnands
Individual
Email: ice@braindump.be

Mankamana Mishra
Cisco Systems, Inc.
821 Alder Drive
Milpitas, CA 95035
United States of America
Email: mankamis@cisco.com

Kamran Raza
Cisco Systems, Inc.
2000 Innovation Drive
Kanata ON K2K-3E8
Canada
Email: skraza@cisco.com

Zhaohui Zhang
Juniper Networks
10 Technology Park Dr.
Westford, MA 01886
United States of America
Email: zzhang@juniper.net

Arkadiy Gulko
Edward Jones wealth management
United States of America

Email: Arkadiy.gulko@edwardjones.com

ntp
Internet-Draft
Updates: 5905, 5906, 8573, 7822, 7821 (if approved)
Intended status: Standards Track
Expires: 16 June 2024

R. Salz
Akamai Technologies

14 December 2023

Updating the NTP Registries
draft-ietf-ntp-update-registries-13

Abstract

The Network Time Protocol (NTP) and Network Time Security (NTS) documents define a number of assigned number registries, collectively called the NTP registries.

Some registries have wrong values, some registries do not follow current common practice, and some are just right. For the sake of completeness, this document reviews all NTP and NTS registries, and makes updates where necessary.

This document updates RFC 5905, RFC 5906, RFC 8573, RFC 7822, and RFC 7821.

Notes

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

This document is a product of the NTP Working Group (https://dt.ietf.org/wg/ntp). Source for this draft and an issue tracker can be found at https://github.com/richsalz/draft-rsalz-update-registries.

RFC Editor: Please update 'this RFC' to refer to this document, once its RFC number is known, through the document.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Network Time Protocol (NTP) and Network Time Security (NTS) documents define a number of assigned number registries, collectively called the NTP registries. The NTP registries can all be found at https://www.iana.org/assignments/ntp-parameters/ntp-parameters.xhtml (https://www.iana.org/assignments/ntp-parameters/ntp-parameters.xhtml) and the NTS registries can all be found at https://www.iana.org/assignments/nts/nts.xhtml (https://www.iana.org/assignments/nts/nts.xhtml).

Some registries have wrong values, some registries do not follow current common practice, and some are just right. For the sake of completeness, this document reviews all NTP and NTS registries, and makes updates where necessary.

The bulk of this document can be divided into two parts:

* First, each registry, its defining document, and a summary of its syntax is defined.

* Second, the revised format and entries for each registry that is being modified is specified.

2. Existing Registries

This section describes the registries and the rules for them. It is intended to be a short summary of the syntax and registration requirements for each registry. The semantics and protocol processing rules for each registry -- that is, how an implementation acts when sending or receiving any of the fields -- are not described here.

2.1. Reference ID, Kiss-o'-Death

[RFC5905] defined two registries; the Reference ID in Section 7.3, and the Kiss-o'-Death in Section 7.4. Both of these are allowed to be four ASCII characters; padded on the right with all-bits-zero if necessary. Entries that start with 0x58, the ASCII letter uppercase X, are reserved for Private or Experimental Use. Both registries are first-come first-served. The formal request to define the registries is in Section 16.

2.2. Extension Field Types

[RFC5905], Section 7.5 defined the on-the-wire format of extension fields but did not create a registry for them.

[RFC5906], Section 13 mentioned the Extension Field Types registry, and defined it indirectly by defining 30 extensions (10 each for request, response, and error response). It did not provide a formal definition of the columns in the registry. [RFC5906], Section 10 splits the Field Type into four subfields, only for use within the Autokey extensions.

[RFC7821] added a new entry, Checksum Complement, to the Extension Field Types registry.

[RFC7822] clarified the processing rules for Extension Field Types, particularly around the interaction with the Message Authentication Code (MAC) field. NTPv4 packets may contain a MAC, but it appears where one would expect an extension with an extension ID of zero and a length of zero. This document adds a registration for the ID, below.

[RFC8573] changed the cryptography used in the MAC field.

[RFC8915] added four new entries to the Extension Field Types registry.

The following problems exists with the current registry:

* Many of the entries in the Extension Field Types registry have swapped some of the nibbles; 0x1234 is listed as 0x1432 for example. This was due to documentation errors with the original implementation of Autokey. This document marks the erroneous values as reserved, in case there is an implementation that used the registered values instead of what the original implementation used.

* Some values were mistakenly re-used.

2.3. Network Time Security Registries

[RFC8915] defines the NTS protocol. Its registries are listed here for completeness, but no changes to them are specified in this document.

Sections 7.1 through 7.5 (inclusive) added entries to existing registries.

Section 7.6 created a new registry, NTS Key Establishment Record Types, that partitions the assigned numbers into three different registration policies: IETF Review, Specification Required, and Private or Experimental Use.

Section 7.7 created a new registry, NTS Next Protocols, that similarly partitions the assigned numbers.

Section 7.8 created two new registries, NTS Error Codes and NTS Warning Codes. Both registries are also partitioned the same way.

3. Updated Registries

The following general guidelines apply to all registries updated here:

* Every registry reserves a partition for Private or Experimental Use.

* Entries with ASCII fields are now limited to uppercase letters or digits; fields starting with 0x58, the uppercase letter "X", are reserved for Private or Experimental Use.

* The policy for every registry is now Specification Required, as defined in [RFC8126], Section 4.6.

The IESG is requested to choose three designated experts, with two being required to approve a registry change. Guidance for such experts is given below.

Each entry described in the sub-sections below is intended to completely replace the existing entry with the same name.

3.1. Guidance to Designated Experts

The designated experts (DE) should be familiar with [RFC8126], particularly Section 5. As that reference suggests, the DE should ascertain the existence of a suitable specification, and verify that it is publicly available. The DE is also expected to check the clarity of purpose and use of the requested code points.

In addition, the DE is expected to be familiar with this document, specifically the history documented here.

4. IANA Considerations

4.1. NTP Reference Identifier Codes

The registration procedure is changed to Specification Required.

The Note is changed to read as follows:

* Codes beginning with the character "X" are reserved for experimentation and development. IANA cannot assign them.

The columns are defined as follows:

* ID (required): a four-byte value padded on the right with all-bits-zero. Each byte other than padding must be an ASCII uppercase letter or digits.

* Clock source (required): A brief text description of the ID.

* Reference (required): the publication defining the ID.

The existing entries are left unchanged.

4.2. NTP Kiss-o'-Death Codes

The registration procedure is changed to Specification Required.

The Note is changed to read as follows:

* Codes beginning with the character "X" are reserved for experimentation and development. IANA cannot assign them.

The columns are defined as follows:

* ID (required): a four-byte value padded on the right with all-bits-zero. Each byte other than padding must be an ASCII uppercase letter or digits.

* Meaning source (required): A brief text description of the ID.

* Reference (required): the publication defining the ID.

The existing entries are left unchanged.

4.3. NTP Extension Field Types

The registration procedure is changed to Specification Required.

The reference [RFC5906] should be added, if possible.

The following two Notes are added:

* Field Types in the range 0xF000 through 0xFFFF, inclusive, are reserved for experimentation and development. IANA cannot assign them. Both NTS Cookie and Autokey Message Request have the same Field Type; in practice this is not a problem as the field semantics will be determined by other parts of the message.

* The "Reserved for historic reasons" is for differences between the original documentation and implementation of Autokey and marks the erroneous values as reserved, in case there is an implementation that used the registered values instead of what the original implementation used.

The columns are defined as follows:

* Field Type (required): A two-byte value in hexadecimal.

* Meaning (required): A brief text description of the field type.

* Reference (required): the publication defining the field type.

The table is replaced with the following entries. IANA is requested to replace "This RFC" with the actual RFC number once assigned.

Field Type	Meaning	Reference
0x0000	Cryptographic MAC	RFC 5905, This RFC
0x0002	Reserved for historic reasons	This RFC
0x0102	Reserved for historic reasons	This RFC
0x0104	Unique Identifier	RFC 8915, Section 5.3
0x0200	No-Operation Request	RFC 5906
0x0201	Association Message Request	RFC 5906
0x0202	Certificate Message Request	RFC 5906
0x0203	Cookie Message Request	RFC 5906
0x0204	Autokey Message Request	RFC 5906
0x0204	NTS Cookie	RFC 8915, Section 5.4
0x0205	Leapseconds Message Request	RFC 5906
0x0206	Sign Message Request	RFC 5906
0x0207	IFF Identity Message Request	RFC 5906
0x0208	GQ Identity Message Request	RFC 5906
0x0209	MV Identity Message Request	RFC 5906
0x0302	Reserved for historic reasons	This RFC
0x0304	NTS Cookie Placeholder	RFC 8915, Section 5.5
0x0402	Reserved for historic reasons	This RFC
0x0404	NTS Authenticator and	RFC 8915,

	Encrypted Extension Fields	Section 5.6
0x0502	Reserved for historic reasons	This RFC
0x0602	Reserved for historic reasons	This RFC
0x0702	Reserved for historic reasons	This RFC
0x0902	Reserved for historic reasons	This RFC
0x2005	UDP Checksum Complement	RFC 7821
0x8002	Reserved for historic reasons	This RFC
0x8102	Reserved for historic reasons	This RFC
0x8200	No-Operation Response	RFC 5906
0x8201	Association Message Response	RFC 5906
0x8202	Certificate Message Response	RFC 5906
0x8203	Cookie Message Response	RFC 5906
0x8204	Autokey Message Response	RFC 5906
0x8205	Leapseconds Message Response	RFC 5906
0x8206	Sign Message Response	RFC 5906
0x8207	IFF Identity Message Response	RFC 5906
0x8208	GQ Identity Message Response	RFC 5906
0x8209	MV Identity Message Response	RFC 5906
0x8302	Reserved for historic reasons	This RFC
0x8402	Reserved for historic reasons	This RFC
0x8502	Reserved for historic reasons	This RFC
0x8602	Reserved for historic reasons	This RFC
0x8702	Reserved for historic reasons	This RFC
0x8802	Reserved for historic reasons	This RFC

0x8902	Reserved for historic reasons	This RFC
0xC002	Reserved for historic reasons	This RFC
0xC102	Reserved for historic reasons	This RFC
0xC200	No-Operation Error Response	RFC 5906
0xC201	Association Message Error Response	RFC 5906
0xC202	Certificate Message Error Response	RFC 5906
0xC203	Cookie Message Error Response	RFC 5906
0xC204	Autokey Message Error Response	RFC 5906
0xC205	Leapseconds Message Error Response	RFC 5906
0xC206	Sign Message Error Response	RFC 5906
0xC207	IFF Identity Message Error Response	RFC 5906
0xC208	GQ Identity Message Error Response	RFC 5906
0xC209	MV Identity Message Error Response	RFC 5906
0xC302	Reserved for historic reasons	This RFC
0xC402	Reserved for historic reasons	This RFC
0xC502	Reserved for historic reasons	This RFC
0xC602	Reserved for historic reasons	This RFC
0xC702	Reserved for historic reasons	This RFC
0xC802	Reserved for historic reasons	This RFC
0xC902	Reserved for historic reasons	This RFC

Table 1

5. Acknowledgements

The members of the NTP Working Group helped a great deal. Notable contributors include:

* Miroslav Lichvar, Red Hat

* Daniel Franke, formerly at Akamai Technologies

* Danny Mayer, Network Time Foundation

* Michelle Cotton, formerly at IANA

* Tamme Dittrich, Tweede Golf

6. Normative References

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/rfc/rfc5905>.

[RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <https://www.rfc-editor.org/rfc/rfc5906>.

[RFC7821] Mizrahi, T., "UDP Checksum Complement in the Network Time Protocol (NTP)", RFC 7821, DOI 10.17487/RFC7821, March 2016, <https://www.rfc-editor.org/rfc/rfc7821>.

[RFC7822] Mizrahi, T. and D. Mayer, "Network Time Protocol Version 4 (NTPv4) Extension Fields", RFC 7822, DOI 10.17487/RFC7822, March 2016, <https://www.rfc-editor.org/rfc/rfc7822>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8573] Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", RFC 8573, DOI 10.17487/RFC8573, June 2019, <https://www.rfc-editor.org/rfc/rfc8573>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <https://www.rfc-editor.org/rfc/rfc8915>.

Author's Address

Rich Salz
Akamai Technologies
Email: rsalz@akamai.com

Web Authorization Protocol
Internet-Draft
Updates: 6749, 6750, 6819 (if approved)
Intended status: Best Current Practice
Expires: 8 November 2024

T. Lodderstedt
SPRIND
J. Bradley
Yubico
A. Labunets
Independent Researcher
D. Fett
Authlete

7 May 2024

OAuth 2.0 Security Best Current Practice
draft-ietf-oauth-security-topics-27

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the threat model and security advice given in RFC 6749, RFC 6750, and RFC 6819 to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0. Further, it deprecates some modes of operation that are deemed less secure or even insecure.

Discussion Venues

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

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/oauthstuff/draft-ietf-oauth-security-topics.

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/.

Copyright Notice

Table of Contents

1. Introduction

Since its publication in [RFC6749] and [RFC6750], OAuth 2.0 (referred to as simply "OAuth" in the following) has gained massive traction in the market and became the standard for API protection and the basis for federated login using OpenID Connect [OpenID.Core]. While OAuth is used in a variety of scenarios and different kinds of deployments, the following challenges can be observed:

* OAuth implementations are being attacked through known implementation weaknesses and anti-patterns (i.e., well-known patterns that are considered insecure). Although most of these threats are discussed in the OAuth 2.0 Threat Model and Security Considerations [RFC6819], continued exploitation demonstrates a need for more specific recommendations, easier to implement mitigations, and more defense in depth.

* OAuth is being used in environments with higher security requirements than considered initially, such as Open Banking, eHealth, eGovernment, and Electronic Signatures. Those use cases call for stricter guidelines and additional protection.

* OAuth is being used in much more dynamic setups than originally anticipated, creating new challenges with respect to security. Those challenges go beyond the original scope of [RFC6749], [RFC6750], and [RFC6819].

  OAuth initially assumed static relationships between clients, authorization servers, and resource servers. The URLs of the servers were known to the client at deployment time and built an anchor for the trust relationships among those parties. The validation of whether the client is talking to a legitimate server was based on TLS server authentication (see [RFC6819], Section 4.5.4). With the increasing adoption of OAuth, this simple model dissolved and, in several scenarios, was replaced by a dynamic establishment of the relationship between clients on one

side and the authorization and resource servers of a particular deployment on the other side. This way, the same client could be used to access services of different providers (in case of standard APIs, such as e-mail or OpenID Connect) or serve as a front end to a particular tenant in a multi-tenant environment. Extensions of OAuth, such as the OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] and OAuth 2.0 Authorization Server Metadata [RFC8414] were developed to support the use of OAuth in dynamic scenarios.

* Technology has changed. For example, the way browsers treat fragments when redirecting requests has changed, and with it, the implicit grant's underlying security model.

This document provides updated security recommendations to address these challenges. It introduces new requirements beyond those defined in existing specifications such as OAuth 2.0 [RFC6749] and OpenID Connect [OpenID.Core] and deprecates some modes of operation that are deemed less secure or even insecure. However, this document does not supplant the security advice given in [RFC6749], [RFC6750], and [RFC6819], but complements those documents.

Naturally, not all existing ecosystems and implementations are compatible with the new requirements and following the best practices described in this document may break interoperability. Nonetheless, it is RECOMMENDED that implementers upgrade their implementations and ecosystems as soon as feasible.

OAuth 2.1, under developement as [I-D.ietf-oauth-v2-1], will incorporate security recommendations from this document.

1.1. Structure

The remainder of this document is organized as follows: The next section summarizes the most important best practices for every OAuth implementor. Afterwards, the updated OAuth attacker model is presented. Subsequently, a detailed analysis of the threats and implementation issues that can be found in the wild today is given along with a discussion of potential countermeasures.

1.2. Conventions and Terminology

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

This specification uses the terms "access token", "authorization endpoint", "authorization grant", "authorization server", "client", "client identifier" (client ID), "protected resource", "refresh token", "resource owner", "resource server", and "token endpoint" defined by OAuth 2.0 [RFC6749].

An "open redirector" is an endpoint on a web server that forwards a users browser to an arbitrary URI obtained from a query parameter.

2. Best Practices

This section describes the core set of security mechanisms and measures the OAuth working group considers to be best practices at the time of writing. Details about these security mechanisms and measures (including detailed attack descriptions) and requirements for less commonly used options are provided in Section 4.

2.1. Protecting Redirect-Based Flows

When comparing client redirect URIs against pre-registered URIs, authorization servers MUST utilize exact string matching except for port numbers in localhost redirection URIs of native apps (see #iuv_countermeasures). This measure contributes to the prevention of leakage of authorization codes and access tokens (see Section 4.1). It can also help to detect mix-up attacks (see Section 4.4).

Clients and authorization servers MUST NOT expose URLs that forward the user’s browser to arbitrary URIs obtained from a query parameter (open redirectors) as described in Section 4.11. Open redirectors can enable exfiltration of authorization codes and access tokens.

Clients MUST prevent Cross-Site Request Forgery (CSRF). In this context, CSRF refers to requests to the redirection endpoint that do not originate at the authorization server, but a malicious third party (see Section 4.4.1.8. of [RFC6819] for details). Clients that have ensured that the authorization server supports Proof Key for Code Exchange (PKCE, [RFC7636]) MAY rely on the CSRF protection provided by PKCE. In OpenID Connect flows, the nonce parameter provides CSRF protection. Otherwise, one-time use CSRF tokens carried in the state parameter that are securely bound to the user agent MUST be used for CSRF protection (see Section 4.7.1).

When an OAuth client can interact with more than one authorization server, a defense against mix-up attacks (see Section 4.4) is REQUIRED. To this end, clients SHOULD

* use the iss parameter as a countermeasure according to [RFC9207], or

* use an alternative countermeasure based on an iss value in the authorization response (such as the iss Claim in the ID Token in [OpenID.Core] or in [OpenID.JARM] responses), processing it as described in [RFC9207].

In the absence of these options, clients MAY instead use distinct redirect URIs to identify authorization endpoints and token endpoints, as described in Section 4.4.2.

An authorization server that redirects a request potentially containing user credentials MUST avoid forwarding these user credentials accidentally (see Section 4.12 for details).

2.1.1. Authorization Code Grant

Clients MUST prevent authorization code injection attacks (see Section 4.5) and misuse of authorization codes using one of the following options:

* Public clients MUST use PKCE [RFC7636] to this end, as motivated in Section 4.5.3.1.
* For confidential clients, the use of PKCE [RFC7636] is RECOMMENDED, as it provides strong protection against misuse and injection of authorization codes as described in Section 4.5.3.1 and, as a side-effect, prevents CSRF even in the presence of strong attackers as described in Section 4.7.1.
* With additional precautions, described in Section 4.5.3.2, confidential OpenID Connect [OpenID.Core] clients MAY use the nonce parameter and the respective Claim in the ID Token instead.

In any case, the PKCE challenge or OpenID Connect nonce MUST be transaction-specific and securely bound to the client and the user agent in which the transaction was started. Authorization servers are encouraged to make a reasonable effort at detecting and preventing the use of constant PKCE challenge or OpenID Connect nonce values.

Note: Although PKCE was designed as a mechanism to protect native apps, this advice applies to all kinds of OAuth clients, including web applications.

When using PKCE, clients SHOULD use PKCE code challenge methods that do not expose the PKCE verifier in the authorization request. Otherwise, attackers that can read the authorization request (cf. Attacker A4 in Section 3) can break the security provided by PKCE. Currently, S256 is the only such method.

Authorization servers MUST support PKCE [RFC7636].

If a client sends a valid PKCE [RFC7636] code_challenge parameter in the authorization request, the authorization server MUST enforce the correct usage of code_verifier at the token endpoint.

Authorization servers MUST mitigate PKCE Downgrade Attacks by ensuring that a token request containing a code_verifier parameter is accepted only if a code_challenge parameter was present in the authorization request, see Section 4.8.2 for details.

Authorization servers MUST provide a way to detect their support for PKCE. It is RECOMMENDED for authorization servers to publish the element code_challenge_methods_supported in their Authorization Server Metadata ([RFC8414]) containing the supported PKCE challenge methods (which can be used by the client to detect PKCE support). Authorization servers MAY instead provide a deployment-specific way to ensure or determine PKCE support by the authorization server.

2.1.2. Implicit Grant

The implicit grant (response type "token") and other response types causing the authorization server to issue access tokens in the authorization response are vulnerable to access token leakage and access token replay as described in Section 4.1, Section 4.2, Section 4.3, and Section 4.6.

Moreover, no standardized method for sender-constraining exists to bind access tokens to a specific client (as recommended in Section 2.2) when the access tokens are issued in the authorization response. This means that an attacker can use the leaked or stolen access token at a resource endpoint.

In order to avoid these issues, clients SHOULD NOT use the implicit grant (response type "token") or other response types issuing access tokens in the authorization response, unless access token injection in the authorization response is prevented and the aforementioned token leakage vectors are mitigated.

Clients SHOULD instead use the response type code (i.e., authorization code grant type) as specified in Section 2.1.1 or any other response type that causes the authorization server to issue access tokens in the token response, such as the code id_token response type. This allows the authorization server to detect replay attempts by attackers and generally reduces the attack surface since access tokens are not exposed in URLs. It also allows the authorization server to sender-constrain the issued tokens (see next section).

2.2. Token Replay Prevention

2.2.1. Access Tokens

A sender-constrained access token scopes the applicability of an access token to a certain sender. This sender is obliged to demonstrate knowledge of a certain secret as a prerequisite for the acceptance of that token at the recipient (e.g., a resource server).

Authorization and resource servers SHOULD use mechanisms for sender-constraining access tokens, such as Mutual TLS for OAuth 2.0 [RFC8705] or OAuth 2.0 Demonstrating Proof of Possession (DPoP) [RFC9449] (see Section 4.10.1), to prevent misuse of stolen and leaked access tokens.

2.2.2. Refresh Tokens

Refresh tokens for public clients MUST be sender-constrained or use refresh token rotation as described in Section 4.14. [RFC6749] already mandates that refresh tokens for confidential clients can only be used by the client for which they were issued.

2.3. Access Token Privilege Restriction

The privileges associated with an access token SHOULD be restricted to the minimum required for the particular application or use case. This prevents clients from exceeding the privileges authorized by the resource owner. It also prevents users from exceeding their privileges authorized by the respective security policy. Privilege restrictions also help to reduce the impact of access token leakage.

In particular, access tokens SHOULD be audience-restricted to a specific resource server, or, if that is not feasible, to a small set of resource servers. To put this into effect, the authorization server associates the access token with certain resource servers and every resource server is obliged to verify, for every request, whether the access token sent with that request was meant to be used for that particular resource server. If it was not, the resource server MUST refuse to serve the respective request. The aud claim as defined in [RFC9068] MAY be used to audience-restrict access tokens. Clients and authorization servers MAY utilize the parameters scope or resource as specified in [RFC6749] and [RFC8707], respectively, to determine the resource server they want to access.

Additionally, access tokens SHOULD be restricted to certain resources and actions on resource servers or resources. To put this into effect, the authorization server associates the access token with the respective resource and actions and every resource server is obliged

to verify, for every request, whether the access token sent with that request was meant to be used for that particular action on the particular resource. If not, the resource server must refuse to serve the respective request. Clients and authorization servers MAY utilize the parameter scope as specified in [RFC6749] and authorization_details as specified in [RFC9396] to determine those resources and/or actions.

2.4. Resource Owner Password Credentials Grant

The resource owner password credentials grant [RFC6749] MUST NOT be used. This grant type insecurely exposes the credentials of the resource owner to the client. Even if the client is benign, this results in an increased attack surface (credentials can leak in more places than just the authorization server) and users are trained to enter their credentials in places other than the authorization server.

Furthermore, the resource owner password credentials grant is not designed to work with two-factor authentication and authentication processes that require multiple user interaction steps. Authentication with cryptographic credentials (cf. WebCrypto [W3C.WebCrypto], WebAuthn [W3C.WebAuthn]) may be impossible to implement with this grant type, as it is usually bound to a specific web origin.

2.5. Client Authentication

Authorization servers SHOULD enforce client authentication if it is feasible, in the particular deployment, to establish a process for issuance/registration of credentials for clients and ensuring the confidentiality of those credentials.

It is RECOMMENDED to use asymmetric cryptography for client authentication, such as mTLS [RFC8705] or signed JWTs ("Private Key JWT") in accordance with [RFC7521] and [RFC7523] (in [OpenID.Core] defined as the client authentication method private_key_jwt). When asymmetric cryptography for client authentication is used, authorization servers do not need to store sensitive symmetric keys, making these methods more robust against leakage of keys.

2.6. Other Recommendations

The use of OAuth Authorization Server Metadata [RFC8414] can help to improve the security of OAuth deployments:

* It ensures that security features and other new OAuth features can be enabled automatically by compliant software libraries.

* It reduces chances for misconfigurations, for example misconfigured endpoint URLs (that might belong to an attacker) or misconfigured security features.
* It can help to facilitate rotation of cryptographic keys and to ensure cryptographic agility.

It is therefore RECOMMENDED that authorization servers publish OAuth Authorization Server Metadata according to [RFC8414] and that clients make use of this Authorization Server Metadata to configure themselves when available.

Under the conditions described in Section 4.15.1, authorization servers SHOULD NOT allow clients to influence their client_id or any claim that could cause confusion with a genuine resource owner.

It is RECOMMENDED to use end-to-end TLS according to [BCP195] between the client and the resource server. If TLS traffic needs to be terminated at an intermediary, refer to Section 4.13 for further security advice.

Authorization responses MUST NOT be transmitted over unencrypted network connections. To this end, authorization servers MUST NOT allow redirect URIs that use the http scheme except for native clients that use Loopback Interface Redirection as described in [RFC8252], Section 7.3.

If the authorization response is sent with in-browser communication techniques like postMessage [WHATWG.postmessage_api] instead of HTTP redirects, both the initiator and receiver of the in-browser message MUST be strictly verified as described in Section 4.17.

To support browser-based clients, endpoints directly accessed by such clients including the Token Endpoint, Authorization Server Metadata Endpoint, jwks_uri Endpoint, and the Dynamic Client Registration Endpoint MAY support the use of Cross-Origin Resource Sharing (CORS, [WHATWG.CORS]). However, CORS MUST NOT be supported at the Authorization Endpoint, as the client does not access this endpoint directly; instead, the client redirects the user agent to it.

3. The Updated OAuth 2.0 Attacker Model

In [RFC6819], a threat model is laid out that describes the threats against which OAuth deployments must be protected. While doing so, [RFC6819] makes certain assumptions about attackers and their capabilities, i.e., implicitly establishes an attacker model. In the following, this attacker model is made explicit and is updated and expanded to account for the potentially dynamic relationships involving multiple parties (as described in Section 1), to include

new types of attackers and to define the attacker model more clearly.

The goal of this document is to ensure that the authorization of a resource owner (with a user agent) at an authorization server and the subsequent usage of the access token at a resource server is protected, as well as practically possible, at least against the following attackers:

* (A1) Web Attackers that can set up and operate an arbitrary number of network endpoints (besides the "honest" ones) including browsers and servers. Web attackers may set up web sites that are visited by the resource owner, operate their own user agents, and participate in the protocol.

  Web attackers may, in particular, operate OAuth clients that are registered at the authorization server, and operate their own authorization and resource servers that can be used (in parallel to the "honest" ones) by the resource owner and other resource owners.

  It must also be assumed that web attackers can lure the user to navigate their browser to arbitrary attacker-chosen URIs at any time. In practice, this can be achieved in many ways, for example, by injecting malicious advertisements into advertisement networks, or by sending legitimate-looking emails.

  Web attackers can use their own user credentials to create new messages as well as any secrets they learned previously. For example, if a web attacker learns an authorization code of a user through a misconfigured redirect URI, the web attacker can then try to redeem that code for an access token.

  They cannot, however, read or manipulate messages that are not targeted towards them (e.g., sent to a URL controlled by a non-attacker controlled authorization server).

* (A2) Network Attackers that additionally have full control over the network over which protocol participants communicate. They can eavesdrop on, manipulate, and spoof messages, except when these are properly protected by cryptographic methods (e.g., TLS). Network attackers can also block arbitrary messages.

While an example for a web attacker would be a customer of an internet service provider, network attackers could be the internet service provider itself, an attacker in a public (Wi-Fi) network using ARP spoofing, or a state-sponsored attacker with access to internet exchange points, for instance.

The aforementioned attackers (A1) and (A2) conform to the attacker model that was used in formal analysis efforts for OAuth [arXiv.1601.01229]. This is a minimal attacker model. Implementers MUST take into account all possible types of attackers in the environment of their OAuth implementations. For example, in [arXiv.1901.11520], a very strong attacker model is used that includes attackers that have full control over the token endpoint. This models effects of a possible misconfiguration of endpoints in the ecosystem, which can be avoided by using authorization server metadata as described in Section 2.6. Such an attacker is therefore not listed here.

However, previous attacks on OAuth have shown that the following types of attackers are relevant in particular:

* (A3) Attackers that can read, but not modify, the contents of the authorization response (i.e., the authorization response can leak to an attacker).

  Examples for such attacks include open redirector attacks, insufficient checking of redirect URIs (see Section 4.1), problems existing on mobile operating systems (where different apps can register themselves on the same URI), mix-up attacks (see Section 4.4), where the client is tricked into sending credentials to an attacker-controlled authorization server, and the fact that URLs are often stored/logged by browsers (history), proxy servers, and operating systems.

* (A4) Attackers that can read, but not modify, the contents of the authorization request (i.e., the authorization request can leak, in the same manner as above, to an attacker).

* (A5) Attackers that can acquire an access token issued by an authorization server. For example, a resource server can be compromised by an attacker, an access token may be sent to an attacker-controlled resource server due to a misconfiguration, or a resource owner is social-engineered into using an attacker-controlled resource server. Also see Section 4.9.2.

(A3), (A4) and (A5) typically occur together with either (A1) or (A2). Attackers can collaborate to reach a common goal.

Note that an attacker (A1) or (A2) can be a resource owner or act as one. For example, such an attacker can use their own browser to replay tokens or authorization codes obtained by any of the attacks described above at the client or resource server.

This document focuses on threats resulting from attackers (A1) to (A5).

4. Attacks and Mitigations

This section gives a detailed description of attacks on OAuth implementations, along with potential countermeasures. Attacks and mitigations already covered in [RFC6819] are not listed here, except where new recommendations are made.

This section further defines additional requirements beyond those defined in Section 2 for certain cases and protocol options.

4.1. Insufficient Redirect URI Validation

Some authorization servers allow clients to register redirect URI patterns instead of complete redirect URIs. The authorization servers then match the redirect URI parameter value at the authorization endpoint against the registered patterns at runtime. This approach allows clients to encode transaction state into additional redirect URI parameters or to register a single pattern for multiple redirect URIs.

This approach turned out to be more complex to implement and more error-prone to manage than exact redirect URI matching. Several successful attacks exploiting flaws in the pattern-matching implementation or concrete configurations have been observed in the wild (see, e.g., [research.rub2]). Insufficient validation of the redirect URI effectively breaks client identification or authentication (depending on grant and client type) and allows the attacker to obtain an authorization code or access token, either

* by directly sending the user agent to a URI under the attacker's control, or
* by exposing the OAuth credentials to an attacker by utilizing an open redirector at the client in conjunction with the way user agents handle URL fragments.

These attacks are shown in detail in the following subsections.

4.1.1. Redirect URI Validation Attacks on Authorization Code Grant

For a client using the grant type code, an attack may work as follows:

Assume the redirect URL pattern https://*.somesite.example/* is registered for the client with the client ID s6BhdRkqt3. The intention is to allow any subdomain of somesite.example to be a valid

redirect URI for the client, for example https://app1.somesite.example/redirect. A naive implementation on the authorization server, however, might interpret the wildcard * as "any character" and not "any character valid for a domain name". The authorization server, therefore, might permit https://attacker.example/.somesite.example as a redirect URI, although attacker.example is a different domain potentially controlled by a malicious party.

The attack can then be conducted as follows:

To begin, the attacker needs to trick the user into opening a tampered URL in their browser that launches a page under the attacker's control, say https://www.evil.example (see Attacker A1 in Section 3).

This URL initiates the following authorization request with the client ID of a legitimate client to the authorization endpoint (line breaks for display only):

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=9ad67f13
     &redirect_uri=https%3A%2F%2Fattacker.example%2F.somesite.example
     HTTP/1.1
Host: server.somesite.example
```

The authorization server validates the redirect URI and compares it to the registered redirect URL patterns for the client s6BhdRkqt3. The authorization request is processed and presented to the user.

If the user does not see the redirect URI or does not recognize the attack, the code is issued and immediately sent to the attacker's domain. If an automatic approval of the authorization is enabled (which is not recommended for public clients according to [RFC6749]), the attack can be performed even without user interaction.

If the attacker impersonates a public client, the attacker can exchange the code for tokens at the respective token endpoint.

This attack will not work as easily for confidential clients, since the code exchange requires authentication with the legitimate client's secret. The attacker can, however, use the legitimate confidential client to redeem the code by performing an authorization code injection attack, see Section 4.5.

It is important to note that redirect URI validation vulnerabilities can also exist if the authorization server handles wildcards properly. For example, assume that the client registers the redirect URL pattern https://*.somesite.example/* and the authorization server

interprets this as "allow redirect URIs pointing to any host residing in the domain somesite.example". If an attacker manages to establish a host or subdomain in somesite.example, he can impersonate the legitimate client. This could be caused, for example, by a subdomain takeover attack [research.udel], where an outdated CNAME record (say, external-service.somesite.example) points to an external DNS name that does no longer exist (say, customer-abc.service.example) and can be taken over by an attacker (e.g., by registering as customer-abc with the external service).

4.1.2. Redirect URI Validation Attacks on Implicit Grant

The attack described above works for the implicit grant as well. If the attacker is able to send the authorization response to an attacker-controlled URI, the attacker will directly get access to the fragment carrying the access token.

Additionally, implicit grants (and also other grants when using response_mode=fragment as defined in [OAuth.Responses]) can be subject to a further kind of attack. It utilizes the fact that user agents re-attach fragments to the destination URL of a redirect if the location header does not contain a fragment (see [RFC9110], Section 17.11). The attack described here combines this behavior with the client as an open redirector (see Section 4.11.1) in order to obtain access tokens. This allows circumvention even of very narrow redirect URI patterns, but not strict URL matching.

Assume the registered URL pattern for client s6BhdRkqt3 is https://client.somesite.example/cb?*, i.e., any parameter is allowed for redirects to https://client.somesite.example/cb. Unfortunately, the client exposes an open redirector. This endpoint supports a parameter redirect_to which takes a target URL and will send the browser to this URL using an HTTP Location header redirect 303.

The attack can now be conducted as follows:

To begin, as above, the attacker needs to trick the user into opening a tampered URL in their browser that launches a page under the attacker’s control, say https://www.evil.example.

Afterwards, the website initiates an authorization request that is very similar to the one in the attack on the code flow. Different to above, it utilizes the open redirector by encoding redirect_to=https://attacker.example into the parameters of the redirect URI and it uses the response type "token" (line breaks for display only):

```
GET /authorize?response_type=token&state=9ad67f13
    &client_id=s6BhdRkqt3
    &redirect_uri=https%3A%2F%2Fclient.somesite.example
     %2Fcb%26redirect_to%253Dhttps%253A%252F
     %252Fattacker.example%252F HTTP/1.1
Host: server.somesite.example
```

Now, since the redirect URI matches the registered pattern, the authorization server permits the request and sends the resulting access token in a 303 redirect (some response parameters omitted for readability):

```
HTTP/1.1 303 See Other
Location: https://client.somesite.example/cb?
          redirect_to%3Dhttps%3A%2F%2Fattacker.example%2Fcb
          #access_token=2YotnFZFEjr1zCsicMWpAA&...
```

At client.somesite.example, the request arrives at the open redirector. The endpoint will read the redirect parameter and will issue an HTTP 303 Location header redirect to the URL https://attacker.example/.

```
HTTP/1.1 303 See Other
Location: https://attacker.example/
```

Since the redirector at client.somesite.example does not include a fragment in the Location header, the user agent will re-attach the original fragment #access_token=2YotnFZFEjr1zCsicMWpAA&... to the URL and will navigate to the following URL:

```
https://attacker.example/#access_token=2YotnFZFEjr1z...
```

The attacker's page at attacker.example can now access the fragment and obtain the access token.

4.1.3. Countermeasures

The complexity of implementing and managing pattern matching correctly obviously causes security issues. This document therefore advises simplifying the required logic and configuration by using exact redirect URI matching. This means the authorization server MUST ensure that the two URIs are equal, see [RFC3986], Section 6.2.1, Simple String Comparison, for details. The only exception is native apps using a localhost URI: In this case, the authorization server MUST allow variable port numbers as described in [RFC8252], Section 7.3.

Additional recommendations:

* Web servers on which redirect URIs are hosted MUST NOT expose open redirectors (see Section 4.11).
* Browsers reattach URL fragments to Location redirection URLs only if the URL in the Location header does not already contain a fragment. Therefore, servers MAY prevent browsers from reattaching fragments to redirection URLs by attaching an arbitrary fragment identifier, for example #_, to URLs in Location headers.
* Clients SHOULD use the authorization code response type instead of response types causing access token issuance at the authorization endpoint. This offers countermeasures against the reuse of leaked credentials through the exchange process with the authorization server and token replay through sender-constraining of the access tokens.

If the origin and integrity of the authorization request containing the redirect URI can be verified, for example when using [RFC9101] or [RFC9126] with client authentication, the authorization server MAY trust the redirect URI without further checks.

4.2. Credential Leakage via Referer Headers

The contents of the authorization request URI or the authorization response URI can unintentionally be disclosed to attackers through the Referer HTTP header (see [RFC9110], Section 10.1.3), by leaking either from the authorization server's or the client's website, respectively. Most importantly, authorization codes or state values can be disclosed in this way. Although specified otherwise in [RFC9110], Section 10.1.3, the same may happen to access tokens conveyed in URI fragments due to browser implementation issues, as illustrated by a (now fixed) issue in the Chromium project [bug.chromium].

4.2.1. Leakage from the OAuth Client

Leakage from the OAuth client requires that the client, as a result of a successful authorization request, renders a page that

* contains links to other pages under the attacker's control and a user clicks on such a link, or
* includes third-party content (advertisements in iframes, images, etc.), for example, if the page contains user-generated content (blog).

As soon as the browser navigates to the attacker's page or loads the third-party content, the attacker receives the authorization response URL and can extract code or state (and potentially access token).

4.2.2. Leakage from the Authorization Server

In a similar way, an attacker can learn state from the authorization request if the authorization endpoint at the authorization server contains links or third-party content as above.

4.2.3. Consequences

An attacker that learns a valid code or access token through a Referer header can perform the attacks as described in Section 4.1.1, Section 4.5, and Section 4.6. If the attacker learns state, the CSRF protection achieved by using state is lost, resulting in CSRF attacks as described in [RFC6819], Section 4.4.1.8.

4.2.4. Countermeasures

The page rendered as a result of the OAuth authorization response and the authorization endpoint SHOULD NOT include third-party resources or links to external sites.

The following measures further reduce the chances of a successful attack:

* Suppress the Referer header by applying an appropriate Referrer Policy [W3C.webappsec-referrer-policy] to the document (either as part of the "referrer" meta attribute or by setting a Referrer-Policy header). For example, the header Referrer-Policy: no-referrer in the response completely suppresses the Referer header in all requests originating from the resulting document.

* Use authorization code instead of response types causing access token issuance from the authorization endpoint.

* Bind the authorization code to a confidential client or PKCE challenge. In this case, the attacker lacks the secret to request the code exchange.

* As described in [RFC6749], Section 4.1.2, authorization codes MUST be invalidated by the authorization server after their first use at the token endpoint. For example, if an authorization server invalidated the code after the legitimate client redeemed it, the attacker would fail to exchange this code later.

  This does not mitigate the attack if the attacker manages to exchange the code for a token before the legitimate client does so. Therefore, [RFC6749] further recommends that, when an attempt is made to redeem a code twice, the authorization server SHOULD revoke all tokens issued previously based on that code.

* The state value SHOULD be invalidated by the client after its first use at the redirection endpoint. If this is implemented, and an attacker receives a token through the Referer header from the client's website, the state was already used, invalidated by the client and cannot be used again by the attacker. (This does not help if the state leaks from the authorization server's website, since then the state has not been used at the redirection endpoint at the client yet.)

* Use the form post response mode instead of a redirect for the authorization response (see [OAuth.Post]).

4.3. Credential Leakage via Browser History

Authorization codes and access tokens can end up in the browser's history of visited URLs, enabling the attacks described in the following.

4.3.1. Authorization Code in Browser History

When a browser navigates to client.example/redirection_endpoint?code=abcd as a result of a redirect from a provider's authorization endpoint, the URL including the authorization code may end up in the browser's history. An attacker with access to the device could obtain the code and try to replay it.

Countermeasures:

* Authorization code replay prevention as described in [RFC6819], Section 4.4.1.1, and Section 4.5.
* Use form post response mode instead of redirect for the authorization response (see [OAuth.Post]).

4.3.2. Access Token in Browser History

An access token may end up in the browser history if a client or a web site that already has a token deliberately navigates to a page like provider.com/get_user_profile?access_token=abcdef. [RFC6750] discourages this practice and advises transferring tokens via a header, but in practice web sites often pass access tokens in query parameters.

In the case of implicit grant, a URL like client.example/redirection_endpoint#access_token=abcdef may also end up in the browser history as a result of a redirect from a provider's authorization endpoint.

Countermeasures:

* Clients MUST NOT pass access tokens in a URI query parameter in the way described in Section 2.3 of [RFC6750]. The authorization code grant or alternative OAuth response modes like the form post response mode [OAuth.Post] can be used to this end.

4.4. Mix-Up Attacks

Mix-up is an attack on scenarios where an OAuth client interacts with two or more authorization servers and at least one authorization server is under the control of the attacker. This can be the case, for example, if the attacker uses dynamic registration to register the client at their own authorization server or if an authorization server becomes compromised.

The goal of the attack is to obtain an authorization code or an access token for an uncompromised authorization server. This is achieved by tricking the client into sending those credentials to the compromised authorization server (the attacker) instead of using them at the respective endpoint of the uncompromised authorization/ resource server.

4.4.1. Attack Description

The description here follows [arXiv.1601.01229], with variants of the attack outlined below.

Preconditions: For this variant of the attack to work, it is assumed that

* the implicit or authorization code grant is used with multiple authorization servers of which one is considered "honest" (H-AS) and one is operated by the attacker (A-AS), and
* the client stores the authorization server chosen by the user in a session bound to the user's browser and uses the same redirection endpoint URI for each authorization server.

In the following, it is further assumed that the client is registered with H-AS (URI: https://honest.as.example, client ID: 7ZGZldHQ) and with A-AS (URI: https://attacker.example, client ID: 666RVZJTA). URLs shown in the following example are shortened for presentation to only include parameters relevant to the attack.

Attack on the authorization code grant:

1. The user selects to start the grant using A-AS (e.g., by clicking on a button on the client's website).

2. The client stores in the user's session that the user selected "A-AS" and redirects the user to A-AS's authorization endpoint with a Location header containing the URL https://attacker.example/authorize?response_type=code&client_id=666RVZJTA.

3. When the user's browser navigates to the attacker's authorization endpoint, the attacker immediately redirects the browser to the authorization endpoint of H-AS. In the authorization request, the attacker replaces the client ID of the client at A-AS with the client's ID at H-AS. Therefore, the browser receives a redirection (303 See Other) with a Location header pointing to https://honest.as.example/authorize?response_type=code&client_id=7ZGZldHQ

4. The user authorizes the client to access their resources at H-AS. (Note that a vigilant user might at this point detect that they intended to use A-AS instead of H-AS. The first attack variant listed below avoids this.) H-AS issues a code and sends it (via the browser) back to the client.

5. Since the client still assumes that the code was issued by A-AS, it will try to redeem the code at A-AS's token endpoint.

6. The attacker therefore obtains code and can either exchange the code for an access token (for public clients) or perform an authorization code injection attack as described in Section 4.5.

Variants:

* Mix-Up With Interception: This variant works only if the attacker can intercept and manipulate the first request/response pair from a user's browser to the client (in which the user selects a certain authorization server and is then redirected by the client to that authorization server), as in Attacker A2 (see Section 3). This capability can, for example, be the result of a man-in-the-middle attack on the user's connection to the client. In the attack, the user starts the flow with H-AS. The attacker intercepts this request and changes the user's selection to A-AS. The rest of the attack proceeds as in Steps 2 and following above.
* Implicit Grant: In the implicit grant, the attacker receives an access token instead of the code in Step 4. The attacker's authorization server receives the access token when the client makes a request to the A-AS userinfo endpoint, or since the client believes it has completed the flow with A-AS, a request to the attacker's resource server.

* Per-AS Redirect URIs: If clients use different redirect URIs for different authorization servers, do not store the selected authorization server in the user's session, and authorization servers do not check the redirect URIs properly, attackers can mount an attack called "Cross-Social Network Request Forgery". These attacks have been observed in practice. Refer to [research.jcs_14] for details.
* OpenID Connect: Some variants can be used to attack OpenID Connect. In these attacks, the attacker misuses features of the OpenID Connect Discovery [OpenID.Discovery] mechanism or replays access tokens or ID Tokens to conduct a mix-up attack. The attacks are described in detail in [arXiv.1704.08539], Appendix A, and [arXiv.1508.04324v2], Section 6 ("Malicious Endpoints Attacks").

4.4.2. Countermeasures

When an OAuth client can only interact with one authorization server, a mix-up defense is not required. In scenarios where an OAuth client interacts with two or more authorization servers, however, clients MUST prevent mix-up attacks. Two different methods are discussed in the following.

For both defenses, clients MUST store, for each authorization request, the issuer they sent the authorization request to and bind this information to the user agent. The issuer serves, via the associated metadata, as an abstract identifier for the combination of the authorization endpoint and token endpoint that are to be used in the flow. If an issuer identifier is not available, for example, if neither OAuth Authorization Server Metadata [RFC8414] nor OpenID Connect Discovery [OpenID.Discovery] is used, a different unique identifier for this tuple or the tuple itself can be used instead. For brevity of presentation, such a deployment-specific identifier will be subsumed under the issuer (or issuer identifier) in the following.

It is important to note that just storing the authorization server URL is not sufficient to identify mix-up attacks. An attacker might declare an uncompromised authorization server's authorization endpoint URL as "their" authorization server URL, but declare a token endpoint under their own control.

4.4.2.1. Mix-Up Defense via Issuer Identification

This defense requires that the authorization server sends its issuer identifier in the authorization response to the client. When receiving the authorization response, the client MUST compare the received issuer identifier to the stored issuer identifier. If there is a mismatch, the client MUST abort the interaction.

There are different ways this issuer identifier can be transported to the client:

* The issuer information can be transported, for example, via a separate response parameter iss, defined in [RFC9207].
* When OpenID Connect is used and an ID Token is returned in the authorization response, the client can evaluate the iss claim in the ID Token.

In both cases, the iss value MUST be evaluated according to [RFC9207].

While this defense may require deploying new OAuth features to transport the issuer information, it is a robust and relatively simple defense against mix-up.

4.4.2.2. Mix-Up Defense via Distinct Redirect URIs

For this defense, clients MUST use a distinct redirect URI for each issuer they interact with.

Clients MUST check that the authorization response was received from the correct issuer by comparing the distinct redirect URI for the issuer to the URI where the authorization response was received on. If there is a mismatch, the client MUST abort the flow.

While this defense builds upon existing OAuth functionality, it cannot be used in scenarios where clients only register once for the use of many different issuers (as in some open banking schemes) and due to the tight integration with the client registration, it is harder to deploy automatically.

Furthermore, an attacker might be able to circumvent the protection offered by this defense by registering a new client with the "honest" authorization server using the redirect URI that the client assigned to the attacker’s authorization server. The attacker could then run the attack as described above, replacing the client ID with the client ID of their newly created client.

This defense SHOULD therefore only be used if other options are not available.

4.5. Authorization Code Injection

An attacker who has gained access to an authorization code contained in an authorization response (see Attacker A3 in Section 3) can try to redeem the authorization code for an access token or otherwise make use of the authorization code.

In the case that the authorization code was created for a public client, the attacker can send the authorization code to the token endpoint of the authorization server and thereby get an access token. This attack was described in Section 4.4.1.1 of [RFC6819].

For confidential clients, or in some special situations, the attacker can execute an authorization code injection attack, as described in the following.

In an authorization code injection attack, the attacker attempts to inject a stolen authorization code into the attacker's own session with the client. The aim is to associate the attacker's session at the client with the victim's resources or identity, thereby giving the attacker at least limited access to the victim's resources.

Besides circumventing the client authentication of confidential clients, other use cases for this attack include:

* The attacker wants to access certain functions in this particular client. As an example, the attacker wants to impersonate their victim in a certain app or on a certain website.
* The authorization or resource servers are limited to certain networks that the attacker is unable to access directly.

Except in these special cases, authorization code injection is usually not interesting when the code is created for a public client, as sending the code to the token endpoint is a simpler and more powerful attack, as described above.

4.5.1. Attack Description

The authorization code injection attack works as follows:

1. The attacker obtains an authorization code (see attacker A3 in Section 3). For the rest of the attack, only the capabilities of a web attacker (A1) are required.
2. From the attacker's device, the attacker starts a regular OAuth authorization process with the legitimate client.

3. In the response of the authorization server to the legitimate client, the attacker replaces the newly created authorization code with the stolen authorization code. Since this response is passing through the attacker's device, the attacker can use any tool that can intercept and manipulate the authorization response to this end. The attacker does not need to control the network.
4. The legitimate client sends the code to the authorization server's token endpoint, along with the redirect_uri and the client's client ID and client secret (or other means of client authentication).
5. The authorization server checks the client secret, whether the code was issued to the particular client, and whether the actual redirect URI matches the redirect_uri parameter (see [RFC6749]).
6. All checks succeed and the authorization server issues access and other tokens to the client. The attacker has now associated their session with the legitimate client with the victim's resources and/or identity.

4.5.2. Discussion

Obviously, the check-in step (5.) will fail if the code was issued to another client ID, e.g., a client set up by the attacker. The check will also fail if the authorization code was already redeemed by the legitimate user and was one-time use only.

An attempt to inject a code obtained via a manipulated redirect URI should also be detected if the authorization server stored the complete redirect URI used in the authorization request and compares it with the redirect_uri parameter.

[RFC6749], Section 4.1.3, requires the authorization server to "... ensure that the redirect_uri parameter is present if the redirect_uri parameter was included in the initial authorization request as described in Section 4.1.1, and if included ensure that their values are identical.". In the attack scenario described above, the legitimate client would use the correct redirect URI it always uses for authorization requests. But this URI would not match the tampered redirect URI used by the attacker (otherwise, the redirect would not land at the attacker's page). So the authorization server would detect the attack and refuse to exchange the code.

This check could also detect attempts to inject an authorization code that had been obtained from another instance of the same client on another device if certain conditions are fulfilled:

* the redirect URI itself needs to contain a nonce or another kind of one-time use, secret data and

* the client has bound this data to this particular instance of the client.

But this approach conflicts with the idea of enforcing exact redirect URI matching at the authorization endpoint. Moreover, it has been observed that providers very often ignore the redirect_uri check requirement at this stage, maybe because it doesn't seem to be security-critical from reading the specification.

Other providers just pattern match the redirect_uri parameter against the registered redirect URI pattern. This saves the authorization server from storing the link between the actual redirect URI and the respective authorization code for every transaction. But this kind of check obviously does not fulfill the intent of the specification, since the tampered redirect URI is not considered. So any attempt to inject an authorization code obtained using the client_id of a legitimate client or by utilizing the legitimate client on another device will not be detected in the respective deployments.

It is also assumed that the requirements defined in [RFC6749], Section 4.1.3, increase client implementation complexity as clients need to store or re-construct the correct redirect URI for the call to the token endpoint.

Asymmetric methods for client authentication do not stop this attack, as the legitimate client authenticates at the token endpoint.

This document therefore recommends instead binding every authorization code to a certain client instance on a certain device (or in a certain user agent) in the context of a certain transaction using one of the mechanisms described next.

4.5.3. Countermeasures

There are two good technical solutions to binding authorization codes to client instances, outlined in the following.

4.5.3.1. PKCE

The PKCE mechanism specified in [RFC7636] can be used as a countermeasure (even though it was originally designed to secure native apps). When the attacker attempts to inject an authorization code, the check of the code_verifier fails: the client uses its correct verifier, but the code is associated with a code_challenge that does not match this verifier.

PKCE does not only protect against the authorization code injection attack but also protects authorization codes created for public clients: PKCE ensures that an attacker cannot redeem a stolen authorization code at the token endpoint of the authorization server without knowledge of the code_verifier.

4.5.3.2. Nonce

OpenID Connect's existing nonce parameter can protect against authorization code injection attacks. The nonce value is one-time use and is created by the client. The client is supposed to bind it to the user agent session and send it with the initial request to the OpenID Provider (OP). The OP puts the received nonce value into the ID Token that is issued as part of the code exchange at the token endpoint. If an attacker injects an authorization code in the authorization response, the nonce value in the client session and the nonce value in the ID token will not match and the attack is detected. The assumption is that an attacker cannot get hold of the user agent state on the victim's device (from which the attacker has stolen the respective authorization code).

It is important to note that this countermeasure only works if the client properly checks the nonce parameter in the ID Token and does not use any issued token until this check has succeeded. More precisely, a client protecting itself against code injection using the nonce parameter

1. MUST validate the nonce in the ID Token obtained from the token endpoint, even if another ID Token was obtained from the authorization response (e.g., response_type=code+id_token), and
2. MUST ensure that, unless and until that check succeeds, all tokens (ID Tokens and the access token) are disregarded and not used for any other purpose.

It is important to note that nonce does not protect authorization codes of public clients, as an attacker does not need to execute an authorization code injection attack. Instead, an attacker can directly call the token endpoint with the stolen authorization code.

4.5.3.3. Other Solutions

Other solutions, like binding state to the code, sender-constraining the code using cryptographic means, or per-instance client credentials are conceivable, but lack support and bring new security requirements.

PKCE is the most obvious solution for OAuth clients as it is available today, while nonce is appropriate for OpenID Connect clients.

4.5.4. Limitations

An attacker can circumvent the countermeasures described above if he can modify the nonce or code_challenge values that are used in the victim's authorization request. The attacker can modify these values to be the same ones as those chosen by the client in their own session in Step 2 of the attack above. (This requires that the victim's session with the client begins after the attacker started their session with the client.) If the attacker is then able to capture the authorization code from the victim, the attacker will be able to inject the stolen code in Step 3 even if PKCE or nonce are used.

This attack is complex and requires a close interaction between the attacker and the victim's session. Nonetheless, measures to prevent attackers from reading the contents of the authorization response still need to be taken, as described in Section 4.1, Section 4.2, Section 4.3, Section 4.4, and Section 4.11.

4.6. Access Token Injection

In an access token injection attack, the attacker attempts to inject a stolen access token into a legitimate client (that is not under the attacker's control). This will typically happen if the attacker wants to utilize a leaked access token to impersonate a user in a certain client.

To conduct the attack, the attacker starts an OAuth flow with the client using the implicit grant and modifies the authorization response by replacing the access token issued by the authorization server or directly making up an authorization server response including the leaked access token. Since the response includes the state value generated by the client for this particular transaction, the client does not treat the response as a CSRF attack and uses the access token injected by the attacker.

4.6.1. Countermeasures

There is no way to detect such an injection attack in pure-OAuth flows since the token is issued without any binding to the transaction or the particular user agent.

In OpenID Connect, the attack can be mitigated, as the authorization response additionally contains an ID Token containing the at_hash claim. The attacker therefore needs to replace both the access token as well as the ID Token in the response. The attacker cannot forge the ID Token, as it is signed or encrypted with authentication. The attacker also cannot inject a leaked ID Token matching the stolen access token, as the nonce claim in the leaked ID Token will (with a very high probability) contain a different value than the one expected in the authorization response.

Note that further protection, like sender-constrained access tokens, is still required to prevent attackers from using the access token at the resource endpoint directly.

The recommendations in Section 2.1.2 follow from this.

4.7. Cross-Site Request Forgery

An attacker might attempt to inject a request to the redirect URI of the legitimate client on the victim's device, e.g., to cause the client to access resources under the attacker's control. This is a variant of an attack known as Cross-Site Request Forgery (CSRF).

4.7.1. Countermeasures

The traditional countermeasure is that clients pass a random value, also known as a CSRF Token, in the state parameter that links the request to the redirect URI to the user agent session as described. This countermeasure is described in detail in [RFC6819], Section 5.3.5. The same protection is provided by PKCE or the OpenID Connect nonce value.

When using PKCE instead of state or nonce for CSRF protection, it is important to note that:

* Clients MUST ensure that the authorization server supports PKCE before using PKCE for CSRF protection. If an authorization server does not support PKCE, state or nonce MUST be used for CSRF protection.

* If state is used for carrying application state, and the integrity of its contents is a concern, clients MUST protect state against tampering and swapping. This can be achieved by binding the contents of state to the browser session and/or signed/encrypted state values. One example of this is discussed in the now-expired draft [I-D.bradley-oauth-jwt-encoded-state].

The authorization server therefore MUST provide a way to detect their support for PKCE. Using Authorization Server Metadata according to [RFC8414] is RECOMMENDED, but authorization servers MAY instead provide a deployment-specific way to ensure or determine PKCE support.

PKCE provides robust protection against CSRF attacks even in presence of an attacker that can read the authorization response (see Attacker A3 in Section 3). When state is used or an ID Token is returned in the authorization response (e.g., response_type=code+id_token), the attacker either learns the state value and can replay it into the forged authorization response, or can extract the nonce from the ID Token and use it in a new request to the authorization server to mint an ID Token with the same nonce. The new ID Token can then be used for the CSRF attack.

4.8. PKCE Downgrade Attack

An authorization server that supports PKCE but does not make its use mandatory for all flows can be susceptible to a PKCE downgrade attack.

The first prerequisite for this attack is that there is an attacker-controllable flag in the authorization request that enables or disables PKCE for the particular flow. The presence or absence of the code_challenge parameter lends itself for this purpose, i.e., the authorization server enables and enforces PKCE if this parameter is present in the authorization request, but does not enforce PKCE if the parameter is missing.

The second prerequisite for this attack is that the client is not using state at all (e.g., because the client relies on PKCE for CSRF prevention) or that the client is not checking state correctly.

Roughly speaking, this attack is a variant of a CSRF attack. The attacker achieves the same goal as in the attack described in Section 4.7: The attacker injects an authorization code (and with that, an access token) that is bound to the attacker's resources into a session between their victim and the client.

4.8.1. Attack Description

1. The user has started an OAuth session using some client at an authorization server. In the authorization request, the client has set the parameter code_challenge=hash(abc) as the PKCE code challenge (with the hash function and parameter encoding as defined in [RFC7636]). The client is now waiting to receive the authorization response from the user's browser.

2. To conduct the attack, the attacker uses their own device to start an authorization flow with the targeted client. The client now uses another PKCE code challenge, say code_challenge=hash(xyz), in the authorization request. The attacker intercepts the request and removes the entire code_challenge parameter from the request. Since this step is performed on the attacker's device, the attacker has full access to the request contents, for example using browser debug tools.
3. If the authorization server allows for flows without PKCE, it will create a code that is not bound to any PKCE code challenge.
4. The attacker now redirects the user's browser to an authorization response URL that contains the code for the attacker's session with the authorization server.
5. The user's browser sends the authorization code to the client, which will now try to redeem the code for an access token at the authorization server. The client will send code_verifier=abc as the PKCE code verifier in the token request.
6. Since the authorization server sees that this code is not bound to any PKCE code challenge, it will not check the presence or contents of the code_verifier parameter. It will issue an access token that belongs to the attacker's resource to the client under the user's control.

4.8.2. Countermeasures

Using state properly would prevent this attack. However, practice has shown that many OAuth clients do not use or check state properly.

Therefore, authorization servers MUST mitigate this attack.

Note that from the view of the authorization server, in the attack described above, a code_verifier parameter is received at the token endpoint although no code_challenge parameter was present in the authorization request for the OAuth flow in which the authorization code was issued.

This fact can be used to mitigate this attack. [RFC7636] already mandates that

* an authorization server that supports PKCE MUST check whether a code challenge is contained in the authorization request and bind this information to the code that is issued; and
* when a code arrives at the token endpoint, and there was a code_challenge in the authorization request for which this code was issued, there must be a valid code_verifier in the token request.

Beyond this, to prevent PKCE downgrade attacks, the authorization server MUST ensure that if there was no code_challenge in the authorization request, a request to the token endpoint containing a code_verifier is rejected.

Authorization servers that mandate the use of PKCE in general or for particular clients implicitly implement this security measure.

4.9. Access Token Leakage at the Resource Server

Access tokens can leak from a resource server under certain circumstances.

4.9.1. Access Token Phishing by Counterfeit Resource Server

An attacker may set up their own resource server and trick a client into sending access tokens to it that are valid for other resource servers (see Attackers A1 and A5 in Section 3). If the client sends a valid access token to this counterfeit resource server, the attacker in turn may use that token to access other services on behalf of the resource owner.

This attack assumes the client is not bound to one specific resource server (and its URL) at development time, but client instances are provided with the resource server URL at runtime. This kind of late binding is typical in situations where the client uses a service implementing a standardized API (e.g., for e-mail, calendar, health, or banking) and where the client is configured by a user or administrator for a service that this user or company uses.

4.9.2. Compromised Resource Server

An attacker may compromise a resource server to gain access to the resources of the respective deployment. Such a compromise may range from partial access to the system, e.g., its log files, to full control over the respective server, in which case all controls can be circumvented and all resources can be accessed. The attacker would also be able to obtain other access tokens held on the compromised system that would potentially be valid to access other resource servers.

Preventing server breaches by hardening and monitoring server systems is considered a standard operational procedure and, therefore, out of the scope of this document. This section focuses on the impact of OAuth-related breaches and the replaying of captured access tokens.

4.9.3. Countermeasures

The following measures should be taken into account by implementers in order to cope with access token replay by malicious actors:

* Sender-constrained access tokens, as described in Section 4.10.1, SHOULD be used to prevent the attacker from replaying the access tokens on other resource servers. If an attacker has only partial access to the compromised system, like a read-only access to web server logs, sender-constrained access tokens may also prevent replay on the compromised system.
* Audience restriction as described in Section 4.10.2 SHOULD be used to prevent replay of captured access tokens on other resource servers.
* The resource server MUST treat access tokens like other sensitive secrets and not store or transfer them in plain text.

The first and second recommendations also apply to other scenarios where access tokens leak (see Attacker A5 in Section 3).

4.10. Misuse of Stolen Access Tokens

Access tokens can be stolen by an attacker in various ways, for example, via the attacks described in Section 4.1, Section 4.2, Section 4.3, Section 4.4 and Section 4.9. Some of these attacks can be mitigated by specific security measures, as described in the respective sections. However, in some cases, these measures are not sufficient or are not implemented correctly. Authorization servers therefore SHOULD ensure that access tokens are sender-constrained and audience-restricted as described in the following. Architecture and performance reasons may prevent the use of these measures in some deployments.

4.10.1. Sender-Constrained Access Tokens

As the name suggests, sender-constrained access tokens scope the applicability of an access token to a certain sender. This sender is obliged to demonstrate knowledge of a certain secret as a prerequisite for the acceptance of that token at a resource server.

A typical flow looks like this:

1. The authorization server associates data with the access token that binds this particular token to a certain client. The binding can utilize the client's identity, but in most cases, the authorization server utilizes key material (or data derived from the key material) known to the client.

2. This key material must be distributed somehow. Either the key material already exists before the authorization server creates the binding or the authorization server creates ephemeral keys. The way pre-existing key material is distributed varies among the different approaches. For example, X.509 Certificates can be used, in which case the distribution happens explicitly during the enrollment process. Or the key material is created and distributed at the TLS layer, in which case it might automatically happen during the setup of a TLS connection.
3. The resource server must implement the actual proof of possession check. This is typically done on the application level, often tied to specific material provided by transport layer (e.g., TLS). The resource server must also ensure that a replay of the proof of possession is not possible.

Two methods for sender-constrained access tokens using proof-of-possession have been defined by the OAuth working group and are in use in practice:

* OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens ([RFC8705]): The approach specified in this document allows the use of mutual TLS (mTLS) for both client authentication and sender-constrained access tokens. For the purpose of sender-constrained access tokens, the client is identified towards the resource server by the fingerprint of its public key. During the processing of an access token request, the authorization server obtains the client's public key from the TLS stack and associates its fingerprint with the respective access tokens. The resource server in the same way obtains the public key from the TLS stack and compares its fingerprint with the fingerprint associated with the access token.
* OAuth 2.0 Demonstrating Proof of Possession (DPoP) ([RFC9449]): DPoP outlines an application-level sender-constraining for access and refresh tokens. It uses proof-of-possession based on a public/private key pair and application-level signing. DPoP can be used with public clients and, in the case of confidential clients, can be combined with any client authentication method.

Note that the security of sender-constrained tokens is undermined when an attacker gets access to the token and the key material. This is, in particular, the case for corrupted client software and cross-site scripting attacks (when the client is running in the browser). If the key material is protected in a hardware or software security module or only indirectly accessible (like in a TLS stack), sender-constrained tokens at least protect against the use of the token when the client is offline, i.e., when the security module or interface is not available to the attacker. This applies to access tokens as well as to refresh tokens (see Section 4.14).

4.10.2. Audience-Restricted Access Tokens

Audience restriction essentially restricts access tokens to a particular resource server. The authorization server associates the access token with the particular resource server and the resource server is then supposed to verify the intended audience. If the access token fails the intended audience validation, the resource server refuses to serve the respective request.

In general, audience restriction limits the impact of token leakage. In the case of a counterfeit resource server, it may (as described below) also prevent abuse of the phished access token at the legitimate resource server.

The audience can be expressed using logical names or physical addresses (like URLs). To prevent phishing, it is necessary to use the actual URL the client will send requests to. In the phishing case, this URL will point to the counterfeit resource server. If the attacker tries to use the access token at the legitimate resource server (which has a different URL), the resource server will detect the mismatch (wrong audience) and refuse to serve the request.

In deployments where the authorization server knows the URLs of all resource servers, the authorization server may just refuse to issue access tokens for unknown resource server URLs.

For this to work, the client needs to tell the authorization server the intended resource server. The mechanism in [RFC8707] can be used for this or the information can be encoded in the scope value (Section 3.3 of [RFC6749]).

Instead of the URL, it is also possible to utilize the fingerprint of the resource server's X.509 certificate as the audience value. This variant would also allow detection of an attempt to spoof the legitimate resource server's URL by using a valid TLS certificate obtained from a different CA. It might also be considered a privacy benefit to hide the resource server URL from the authorization server.

Audience restriction may seem easier to use since it does not require any cryptography on the client side. Still, since every access token is bound to a specific resource server, the client also needs to obtain a single resource server-specific access token when accessing several resource servers. (Resource indicators, as specified in [RFC8707], can help to achieve this.) [I-D.ietf-oauth-token-binding] had the same property since different token-binding IDs must be associated with the access token. Using [RFC8705], on the other hand, allows a client to use the access token at multiple resource servers.

It should be noted that audience restrictions, or generally speaking an indication by the client to the authorization server where it wants to use the access token, have additional benefits beyond the scope of token leakage prevention. It allows the authorization server to create a different access token whose format and content are specifically minted for the respective server. This has huge functional and privacy advantages in deployments using structured access tokens.

4.10.3. Discussion: Preventing Leakage via Metadata

An authorization server could provide the client with additional information about the locations where it is safe to use its access tokens. This approach, and why it is not recommended, is discussed in the following.

In the simplest form, this would require the authorization server to publish a list of its known resource servers, illustrated in the following example using a non-standard Authorization Server Metadata parameter resource_servers:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "issuer":"https://server.somesite.example",
  "authorization_endpoint":
    "https://server.somesite.example/authorize",
  "resource_servers":[
    "email.somesite.example",
    "storage.somesite.example",
    "video.somesite.example"
  ]
  ...
}
```

The authorization server could also return the URL(s) an access token is good for in the token response, illustrated by the example and non-standard return parameter access_token_resource_server:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "access_token_resource_server":
    "https://hostedresource.somesite.example/path1",
...
}
```

This mitigation strategy would rely on the client to enforce the security policy and to only send access tokens to legitimate destinations. Results of OAuth-related security research (see for example [research.ubc] and [research.cmu]) indicate a large portion of client implementations do not or fail to properly implement security controls, like state checks. So relying on clients to prevent access token phishing is likely to fail as well. Moreover, given the ratio of clients to authorization and resource servers, it is considered the more viable approach to move as much as possible security-related logic to those entities. Clearly, the client has to contribute to the overall security. However, there are alternative countermeasures, as described before, that provide a better balance between the involved parties.

4.11. Open Redirection

The following attacks can occur when an authorization server or client has an open redirector. Such endpoints are sometimes implemented, for example, to show a message before a user is then redirected to an external website, or to redirect users back to a URL they were intending to visit before being interrupted, e.g., by a login prompt.

4.11.1. Client as Open Redirector

Clients MUST NOT expose open redirectors. Attackers may use open redirectors to produce URLs pointing to the client and utilize them to exfiltrate authorization codes and access tokens, as described in Section 4.1.2. Another abuse case is to produce URLs that appear to point to the client. This might trick users into trusting the URL and following it in their browser. This can be abused for phishing.

In order to prevent open redirection, clients should only redirect if the target URLs are allowed or if the origin and integrity of a request can be authenticated. Countermeasures against open redirection are described by OWASP [owasp.redir].

4.11.2. Authorization Server as Open Redirector

Just as with clients, attackers could try to utilize a user's trust in the authorization server (and its URL in particular) for performing phishing attacks. OAuth authorization servers regularly redirect users to other websites (the clients), but must do so safely.

[RFC6749], Section 4.1.2.1, already prevents open redirects by stating that the authorization server MUST NOT automatically redirect the user agent in case of an invalid combination of client_id and redirect_uri.

However, an attacker could also utilize a correctly registered redirect URI to perform phishing attacks. The attacker could, for example, register a client via dynamic client registration [RFC7591] and execute one of the following attacks:

1. Intentionally send an erroneous authorization request, e.g., by using an invalid scope value, thus instructing the authorization server to redirect the user-agent to its phishing site.
2. Intentionally send a valid authorization request with client_id and redirect_uri controlled by the attacker. After the user authenticates, the authorization server prompts the user to provide consent to the request. If the user notices an issue with the request and declines the request, the authorization server still redirects the user agent to the phishing site. In this case, the user agent will be redirected to the phishing site regardless of the action taken by the user.
3. Intentionally send a valid silent authentication request (prompt=none) with client_id and redirect_uri controlled by the attacker. In this case, the authorization server will automatically redirect the user agent to the phishing site.

The authorization server MUST take precautions to prevent these threats. The authorization server MUST always authenticate the user first and, with the exception of the silent authentication use case, prompt the user for credentials when needed, before redirecting the user. Based on its risk assessment, the authorization server needs to decide whether it can trust the redirect URI or not. It could take into account URI analytics done internally or through some external service to evaluate the credibility and trustworthiness of content behind the URI, and the source of the redirect URI and other client data.

The authorization server SHOULD only automatically redirect the user agent if it trusts the redirect URI. If the URI is not trusted, the authorization server MAY inform the user and rely on the user to make the correct decision.

4.12. 307 Redirect

At the authorization endpoint, a typical protocol flow is that the authorization server prompts the user to enter their credentials in a form that is then submitted (using the HTTP POST method) back to the authorization server. The authorization server checks the credentials and, if successful, redirects the user agent to the client's redirection endpoint.

In [RFC6749], the HTTP status code 302 is used for this purpose, but "any other method available via the user-agent to accomplish this redirection is allowed". When the status code 307 is used for redirection instead, the user agent will send the user's credentials via HTTP POST to the client.

This discloses the sensitive credentials to the client. If the client is malicious, it can use the credentials to impersonate the user at the authorization server.

The behavior might be unexpected for developers but is defined in [RFC9110], Section 15.4.8. This status code does not require the user agent to rewrite the POST request to a GET request and thereby drop the form data in the POST request body.

In the HTTP standard [RFC9110], only the status code 303 unambiguously enforces rewriting the HTTP POST request to an HTTP GET request. For all other status codes, including the popular 302, user agents can opt not to rewrite POST to GET requests and therefore to reveal the user's credentials to the client. (In practice, however, most user agents will only show this behaviour for 307 redirects.)

Authorization servers that redirect a request that potentially contains the user's credentials therefore MUST NOT use the HTTP 307 status code for redirection. If an HTTP redirection (and not, for example, JavaScript) is used for such a request, the authorization server SHOULD use HTTP status code 303 (See Other).

4.13. TLS Terminating Reverse Proxies

A common deployment architecture for HTTP applications is to hide the application server behind a reverse proxy that terminates the TLS connection and dispatches the incoming requests to the respective application server nodes.

This section highlights some attack angles of this deployment architecture with relevance to OAuth and gives recommendations for security controls.

In some situations, the reverse proxy needs to pass security-related data to the upstream application servers for further processing. Examples include the IP address of the request originator, token-binding IDs, and authenticated TLS client certificates. This data is usually passed in HTTP headers added to the upstream request. While the headers are often custom, application-specific headers, standardized header fields for client certificates and client certificate chains are defined in [RFC9440].

If the reverse proxy passes through any header sent from the outside, an attacker could try to directly send the faked header values through the proxy to the application server in order to circumvent security controls that way. For example, it is standard practice of reverse proxies to accept X-Forwarded-For headers and just add the origin of the inbound request (making it a list). Depending on the logic performed in the application server, the attacker could simply add an allowed IP address to the header and render the protection useless.

A reverse proxy MUST therefore sanitize any inbound requests to ensure the authenticity and integrity of all header values relevant for the security of the application servers.

If an attacker were able to get access to the internal network between the proxy and application server, the attacker could also try to circumvent security controls in place. It is, therefore, essential to ensure the authenticity of the communicating entities. Furthermore, the communication link between the reverse proxy and application server MUST be protected against eavesdropping, injection, and replay of messages.

4.14. Refresh Token Protection

Refresh tokens are a convenient and user-friendly way to obtain new access tokens. They also add to the security of OAuth, since they allow the authorization server to issue access tokens with a short lifetime and reduced scope, thus reducing the potential impact of access token leakage.

4.14.1. Discussion

Refresh tokens are an attractive target for attackers since they represent the full scope of grant a resource owner delegated to a certain client and they are not further constrained to a specific resource. If an attacker is able to exfiltrate and successfully replay a refresh token, the attacker will be able to mint access tokens and use them to access resource servers on behalf of the resource owner.

[RFC6749] already provides robust baseline protection by requiring

* confidentiality of the refresh tokens in transit and storage,
* the transmission of refresh tokens over TLS-protected connections between authorization server and client,
* the authorization server to maintain and check the binding of a refresh token to a certain client and authentication of this client during token refresh, if possible, and
* that refresh tokens cannot be generated, modified, or guessed.

[RFC6749] also lays the foundation for further (implementation-specific) security measures, such as refresh token expiration and revocation as well as refresh token rotation by defining respective error codes and response behaviors.

This specification gives recommendations beyond the scope of [RFC6749] and clarifications.

4.14.2. Recommendations

Authorization servers MUST determine, based on a risk assessment, whether to issue refresh tokens to a certain client. If the authorization server decides not to issue refresh tokens, the client MAY obtain a new access token by utilizing other grant types, such as the authorization code grant type. In such a case, the authorization server may utilize cookies and persistent grants to optimize the user experience.

If refresh tokens are issued, those refresh tokens MUST be bound to the scope and resource servers as consented by the resource owner. This is to prevent privilege escalation by the legitimate client and reduce the impact of refresh token leakage.

For confidential clients, [RFC6749] already requires that refresh tokens can only be used by the client for which they were issued.

Authorization servers MUST utilize one of these methods to detect refresh token replay by malicious actors for public clients:

* *Sender-constrained refresh tokens:* the authorization server cryptographically binds the refresh token to a certain client instance, e.g., by utilizing [RFC8705] or [RFC9449].

* *Refresh token rotation:* the authorization server issues a new refresh token with every access token refresh response. The previous refresh token is invalidated but information about the relationship is retained by the authorization server. If a refresh token is compromised and subsequently used by both the attacker and the legitimate client, one of them will present an invalidated refresh token, which will inform the authorization server of the breach. The authorization server cannot determine which party submitted the invalid refresh token, but it will revoke the active refresh token. This stops the attack at the cost of forcing the legitimate client to obtain a fresh authorization grant.

  Implementation note: The grant to which a refresh token belongs may be encoded into the refresh token itself. This can enable an authorization server to efficiently determine the grant to which a refresh token belongs, and by extension, all refresh tokens that need to be revoked. Authorization servers MUST ensure the integrity of the refresh token value in this case, for example, using signatures.

Authorization servers MAY revoke refresh tokens automatically in case of a security event, such as:

* password change
* logout at the authorization server

Refresh tokens SHOULD expire if the client has been inactive for some time, i.e., the refresh token has not been used to obtain fresh access tokens for some time. The expiration time is at the discretion of the authorization server. It might be a global value or determined based on the client policy or the grant associated with the refresh token (and its sensitivity).

4.15. Client Impersonating Resource Owner

Resource servers may make access control decisions based on the identity of a resource owner for which an access token was issued, or based on the identity of a client in the client credentials grant. For example, [RFC9068] (JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens) describes a data structure for access tokens containing a sub claim defined as follows:

> In cases of access tokens obtained through grants where a resource owner is involved, such as the authorization code grant, the value of sub SHOULD correspond to the subject identifier of the resource owner. In cases of access tokens obtained through grants where no resource owner is involved, such as the client credentials grant, the value of sub SHOULD correspond to an identifier the authorization server uses to indicate the client application.

If both options are possible, a resource server may mistake a client's identity for the identity of a resource owner. For example, if a client is able to choose its own client_id during registration with the authorization server, a malicious client may set it to a value identifying a resource owner (e.g., a sub value if OpenID Connect is used). If the resource server cannot properly distinguish between access tokens obtained with involvement of the resource owner and those without, the client may accidentally be able to access resources belonging to the resource owner.

This attack potentially affects not only implementations using [RFC9068], but also similar, bespoke solutions.

4.15.1. Countermeasures

Authorization servers SHOULD NOT allow clients to influence their client_id or any claim that could cause confusion with a genuine resource owner if a common namespace for client IDs and user identifiers exists, such as in the sub claim shown above. Where this cannot be avoided, authorization servers MUST provide other means for the resource server to distinguish between the two types of access tokens.

4.16. Clickjacking

As described in Section 4.4.1.9 of [RFC6819], the authorization request is susceptible to clickjacking attacks, also called user interface redressing. In such an attack, an attacker embeds the authorization endpoint user interface in an innocuous context. A user believing to interact with that context, for example, by clicking on buttons, inadvertently interacts with the authorization

endpoint user interface instead. The opposite can be achieved as well: A user believing to interact with the authorization endpoint might inadvertently type a password into an attacker-provided input field overlaid over the original user interface. Clickjacking attacks can be designed such that users can hardly notice the attack, for example using almost invisible iframes overlaid on top of other elements.

An attacker can use this vector to obtain the user's authentication credentials, change the scope of access granted to the client, and potentially access the user's resources.

Authorization servers MUST prevent clickjacking attacks. Multiple countermeasures are described in [RFC6819], including the use of the X-Frame-Options HTTP response header field and frame-busting JavaScript. In addition to those, authorization servers SHOULD also use Content Security Policy (CSP) level 2 [W3C.CSP-2] or greater.

To be effective, CSP must be used on the authorization endpoint and, if applicable, other endpoints used to authenticate the user and authorize the client (e.g., the device authorization endpoint, login pages, error pages, etc.). This prevents framing by unauthorized origins in user agents that support CSP. The client MAY permit being framed by some other origin than the one used in its redirection endpoint. For this reason, authorization servers SHOULD allow administrators to configure allowed origins for particular clients and/or for clients to register these dynamically.

Using CSP allows authorization servers to specify multiple origins in a single response header field and to constrain these using flexible patterns (see [W3C.CSP-2] for details). Level 2 of this standard provides a robust mechanism for protecting against clickjacking by using policies that restrict the origin of frames (using frame-ancestors) together with those that restrict the sources of scripts allowed to execute on an HTML page (by using script-src). A non-normative example of such a policy is shown in the following listing:

```
HTTP/1.1 200 OK
Content-Security-Policy: frame-ancestors https://ext.example.org:8000
Content-Security-Policy: script-src 'self'
X-Frame-Options: ALLOW-FROM https://ext.example.org:8000
...
```

Because some user agents do not support [W3C.CSP-2], this technique SHOULD be combined with others, including those described in [RFC6819], unless such legacy user agents are explicitly unsupported by the authorization server. Even in such cases, additional countermeasures SHOULD still be employed.

4.17. Attacks on In-Browser Communication Flows

If the authorization response is sent with in-browser communication techniques like postMessage [WHATWG.postmessage_api] instead of HTTP redirects, messages may inadvertently be sent to malicious origins or injected from malicious origins.

4.17.1. Examples

The following non-normative pseudocode examples of attacks using in-browser communication are described in [research.rub]:

4.17.1.1. Insufficient Limitation of Receiver Origins

When sending the authorization response or token response via postMessage, the authorization server sends the response to the wildcard origin "*" instead of the client's origin. When the window to which the response is sent is controlled by an attacker, the attacker can read the response.

```
window.opener.postMessage(
  {
    code: "ABC",
    state: "123"
  },
  "*" // any website in the opener window can receive the message
)
```

4.17.1.2. Insufficient URI Validation

When sending the authorization response or token response via postMessage, the authorization server may not check the receiver origin against the redirect URI and instead, for example, send the response to an origin provided by an attacker. This is analogous to the attack described in Section 4.1.

```
window.opener.postMessage(
  {
    code: "ABC",
    state: "123"
  },
  "https://attacker.example" // attacker-provided value
)
```

4.17.1.3. Injection after Insufficient Validation of Sender Origin

A client that expects the authorization response or token response via postMessage may not validate the sender origin of the message. This may allow an attacker to inject an authorization response or token response into the client.

In the case of a maliciously injected authorization response, the attack is a variant of the CSRF attacks described in Section 4.7. The countermeasures described in Section 4.7 apply to this attack as well.

In the case of a maliciously injected token response, sender-constrained access tokens as described in Section 4.10.1 may prevent the attack under some circumstances, but additional countermeasures as described next are generally required.

4.17.2. Recommendations

When comparing client receiver origins against pre-registered origins, authorization servers MUST utilize exact string matching as described in Section 4.1.3. Authorization servers MUST send postMessages to trusted client receiver origins, as shown in the following, non-normative example:

```
window.opener.postMessage(
  {
    code: "ABC",
    state: "123"
  },
  "https://client.example" // use explicit client origin
)
```

Wildcard origins like "*" in postMessage MUST NOT be used as attackers can use them to leak a victim's in-browser message to malicious origins. Both measures contribute to the prevention of leakage of authorization codes and access tokens (see Section 4.1).

Clients MUST prevent injection of in-browser messages on the client receiver endpoint. Clients MUST utilize exact string matching to compare the initiator origin of an in-browser message with the authorization server origin, as shown in the following, non-normative example:

```
window.addEventListener("message", (e) => {
  // validate exact authorization server origin
  if (e.origin === "https://honest.as.example") {
    // process e.data.code and e.data.state
  }
})
```

Since in-browser communication flows only apply a different communication technique (i.e., postMessage instead of HTTP redirect), all measures protecting the authorization response listed in Section 2.1 MUST be applied equally.

5. Acknowledgements

We would like to thank Brock Allen, Annabelle Richard Backman, Dominick Baier, Vittorio Bertocci, Brian Campbell, Bruno Crispo, William Dennis, George Fletcher, Matteo Golinelli, Dick Hardt, Joseph Heenan, Pedram Hosseyni, Phil Hunt, Tommaso Innocenti, Louis Jannett, Jared Jennings, Michael B. Jones, Engin Kirda, Konstantin Lapine, Neil Madden, Christian Mainka, Jim Manico, Nov Matake, Doug McDorman, Ali Mirheidari, Vladislav Mladenov, Karsten Meyer zu Selhausen, Kaan Onarioglu, Aaron Parecki, Michael Peck, Johan Peeters, Nat Sakimura, Guido Schmitz, Jörg Schwenk, Rifaat Shekh-Yusef, Travis Spencer, Petteri Stenius, Tomek Stojecki, Tim Wuertele, David Waite and Hans Zandbelt for their valuable feedback.

6. IANA Considerations

This draft makes no requests to IANA.

7. Security Considerations

Security considerations are described in Section 2, Section 3, and Section 4.

8. References

8.1. Normative References

[BCP195] IETF, "BCP195", <https://www.rfc-editor.org/info/bcp195>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/info/rfc7521>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/info/rfc7523>.

[RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/info/rfc8252>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/info/rfc8705>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <https://www.rfc-editor.org/info/rfc9068>.

8.2. Informative References

[I-D.bradley-oauth-jwt-encoded-state]
Bradley, J., Lodderstedt, T., and H. Zandbelt, "Encoding claims in the OAuth 2 state parameter using a JWT", Work

in Progress, Internet-Draft, draft-bradley-oauth-jwt-encoded-state-09, 4 November 2018, <https://datatracker.ietf.org/doc/html/draft-bradley-oauth-jwt-encoded-state-09>.

[I-D.ietf-oauth-token-binding]
Jones, M., Campbell, B., Bradley, J., and W. Denniss, "OAuth 2.0 Token Binding", Work in Progress, Internet-Draft, draft-ietf-oauth-token-binding-08, 19 October 2018, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-token-binding-08>.

[I-D.ietf-oauth-v2-1]
Hardt, D., Parecki, A., and T. Lodderstedt, "The OAuth 2.1 Authorization Framework", Work in Progress, Internet-Draft, draft-ietf-oauth-v2-1-10, 9 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-10>.

[OAuth.Post]
Jones, M. and B. Campbell, "OAuth 2.0 Form Post Response Mode", 27 April 2015, <https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html>.

[OAuth.Responses]
de Medeiros, B., Scurtescu, M., Tarjan, P., and M. Jones, "OAuth 2.0 Multiple Response Type Encoding Practices", 25 February 2014, <https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html>.

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2", 15 December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Discovery]
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 2", 15 December 2023, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[OpenID.JARM]
Lodderstedt, T. and B. Campbell, "Financial-grade API: JWT Secured Authorization Response Mode for OAuth 2.0 (JARM)", 17 October 2018, <https://openid.net/specs/openid-financial-api-jarm.html>.

[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>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/info/rfc7636>.

[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>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/info/rfc8707>.

[RFC9101] Sakimura, N., Bradley, J., and M. Jones, "The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)", RFC 9101, DOI 10.17487/RFC9101, August 2021, <https://www.rfc-editor.org/info/rfc9101>.

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

[RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/info/rfc9126>.

[RFC9207] Meyer zu Selhausen, K. and D. Fett, "OAuth 2.0 Authorization Server Issuer Identification", RFC 9207, DOI 10.17487/RFC9207, March 2022, <https://www.rfc-editor.org/info/rfc9207>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/info/rfc9396>.

[RFC9440] Campbell, B. and M. Bishop, "Client-Cert HTTP Header Field", RFC 9440, DOI 10.17487/RFC9440, July 2023, <https://www.rfc-editor.org/info/rfc9440>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/info/rfc9449>.

[W3C.CSP-2] West, M., Barth, A., and D. Veditz, "Content Security Policy Level 2", July 2015, <https://www.w3.org/TR/CSP2>.

[W3C.WebAuthn] Hodges, J., Jones, J.C., Jones, M.B., Kumar, A., and E. Lundberg, "Web Authentication: An API for accessing Public Key Credentials Level 2", 8 April 2021, <https://www.w3.org/TR/2021/REC-webauthn-2-20210408/>.

[W3C.WebCrypto] Watson, M., "Web Cryptography API", 26 January 2017, <https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126/>.

[W3C.webappsec-referrer-policy] Eisinger, J. and E. Stark, "Referrer Policy", 20 April 2017, <https://w3c.github.io/webappsec-referrer-policy>.

[WHATWG.CORS] "Fetch Standard: CORS protocol", <https://fetch.spec.whatwg.org/#http-cors-protocol>.

[WHATWG.postmessage_api] "HTML Living Standard: Cross-document messaging", <https://html.spec.whatwg.org/multipage/web-messaging.html#web-messaging>.

[arXiv.1508.04324v2] Mladenov, V., Mainka, C., and J. Schwenk, "On the security of modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect", arXiv 1508.04324v2, 7 January 2016, <https://arxiv.org/abs/1508.04324v2/>.

[arXiv.1601.01229] Fett, D., Küsters, R., and G. Schmitz, "A Comprehensive Formal Security Analysis of OAuth 2.0", arXiv 1601.01229, 6 January 2016, <https://arxiv.org/abs/1601.01229/>.

[arXiv.1704.08539]
Fett, D., Küsters, R., and G. Schmitz, "The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines", arXiv 1704.08539, 27 April 2017, <https://arxiv.org/abs/1704.08539/>.

[arXiv.1901.11520]
Fett, D., Hosseyni, P., and R. Küsters, "An Extensive Formal Security Analysis of the OpenID Financial-grade API", arXiv 1901.11520, 31 January 2019, <https://arxiv.org/abs/1901.11520/>.

[bug.chromium]
"Referer header includes URL fragment when opening link using New Tab", <https://issues.chromium.org/issues/40076763/>.

[owasp.redir]
"OWASP Cheat Sheet Series - Unvalidated Redirects and Forwards", <https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html>.

[research.cmu]
Chen, E., Pei, Y., Chen, S., Tian, Y., Kotcher, R., and P. Tague, "OAuth Demystified for Mobile Application Developers", November 2014, <https://css.csail.mit.edu/6.858/2012/readings/oauth-sso.pdf>.

[research.jcs_14]
Bansal, C., Bhargavan, K., Delignat-Lavaud, A., and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis", 23 April 2014, <https://www.doc.ic.ac.uk/~maffeis/papers/jcs14.pdf>.

[research.rub]
Jannett, L., Mladenov, V., Mainka, C., and J. Schwenk, "DISTINCT: Identity Theft using In-Browser Communications in Dual-Window Single Sign-On", DOI 10.1145/3548606.3560692, 7 November 2022, <https://distinct-sso.com/paper.pdf>.

[research.rub2]
Fries, C., "Security Analysis of Real-Life OpenID Connect Implementations", 20 December 2020, <https://www.nds.rub.de/media/ei/arbeiten/2021/05/03/masterthesis.pdf>.

[research.ubc]
Sun, S.-T. and K. Beznosov, "The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems", October 2012, <https://passwordresearch.com/papers/paper267.html>.

[research.udel]
Liu, D., Hao, S., and H. Wang, "All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records", 24 October 2016, <https://www.eecis.udel.edu/~hnw/paper/ccs16a.pdf>.

Appendix A. Document History

[[To be removed from the final specification]]

-27

* Mostly editorial feedback from Microsoft incorporated
* Feedback from SECDIR review incorporated

-26

* Feedback from ARTART review incorporated
* Gen-ART review (typo fixes)

-25

* Shepherd's writeup feedback: Removed discussion on outdated POP approaches
* Shepherd's writeup feedback: Clarify relationship to other document.
* Shepherd's writeup feedback: Expand abbreviations
* Shepherd's writeup feedback: Better explain attacker model
* Shepherd's writeup feedback: Various editorial changes
* AD review: Mention updated documents in abstract
* AD review: Fix HTTP reference
* AD review: Clarification in the attacker model
* AD review: Various editorial and minor changes

-24

* Some feedback from shepherd's writeup incorporated
* Cleaned up references
* Clarification on mix-up attack
* Add researcher names to acknowledgements
* Removed sentence stating that only MTLS is standardized; DPoP is now as well

-23

* Added CORS considerations
* Reworded Section 4.15.1 to be more in line with OAuth 2.1
* Editorial changes
* Clarifications and updated references

-22

* Added section on securing in-browser communication
* Merged section on phishing via AS into existing section on open redirectors
* Restructure and move section on sender-constrained tokens
* Mention RFCs for Private Key JWK method

-21

* Improved wording on phishing via AS

-20

* Improved description of authorization code injection attacks and PKCE protection
* Removed recommendation for MTLS in discussion (not reflected in actual Recommendations section)
* Reworded "placeholder" text in security considerations.
* Alphabetized list of names and fixed unicode problem
* Explained Clickjacking
* Explained Open Redirectors
* Clarified references to attacker model by including a link to Section 3
* Clarified description of "CSRF tokens" and reference to RFC6819
* Described that OIDC can prevent access token injection
* Updated references

-19

* Changed affiliation of Andrey Labunets
* Editorial change to clarify the new recommendations for refresh tokens

-18

* Fix editorial and spelling issues.
* Change wording for disallowing HTTP redirect URIs.

-17

* Make the use of metadata RECOMMENDED for both servers and clients
* Make announcing PKCE support in metadata the RECOMMENDED way (before: either metadata or deployment-specific way)
* AS also MUST NOT expose open redirectors.
* Mention that attackers can collaborate.
* Update recommendations regarding mix-up defense, building upon [RFC9207].
* Improve description of mix-up attack.
* Make HTTPS mandatory for most redirect URIs.

-16

* Make MTLS a suggestion, not RECOMMENDED.
* Add important requirements when using nonce for code injection protection.
* Highlight requirements for refresh token sender-constraining.
* Make PKCE a MUST for public clients.
* Describe PKCE Downgrade Attacks and countermeasures.
* Allow variable port numbers in localhost redirect URIs as in RFC8252, Section 7.3.

-15

* Update reference to DPoP
* Fix reference to RFC8414
* Move to xml2rfcv3

-14

* Added info about using CSP to prevent clickjacking
* Changes from WGLC feedback
* Editorial changes
* AS MUST announce PKCE support either in metadata or using deployment-specific ways (before: SHOULD)

-13

* Discourage use of Resource Owner Password Credentials Grant
* Added text on client impersonating resource owner
* Recommend asymmetric methods for client authentication
* Encourage use of PKCE mode "S256"
* PKCE may replace state for CSRF protection
* AS SHOULD publish PKCE support
* Cleaned up discussion on auth code injection
* AS MUST support PKCE

-12

* Added updated attacker model

-11

* Adapted section 2.1.2 to outcome of consensus call
* more text on refresh token inactivity and implementation note on refresh token replay detection via refresh token rotation

-10

* incorporated feedback by Joseph Heenan
* changed occurrences of SHALL to MUST
* added text on lack of token/cert binding support tokens issued in the authorization response as justification to not recommend issuing tokens there at all
* added requirement to authenticate clients during code exchange (PKCE or client credential) to 2.1.1.
* added section on refresh tokens
* editorial enhancements to 2.1.2 based on feedback

-09

* changed text to recommend not to use implicit but code
* added section on access token injection
* reworked sections 3.1 through 3.3 to be more specific on implicit grant issues

-08

* added recommendations re implicit and token injection
* uppercased key words in Section 2 according to RFC 2119

-07

* incorporated findings of Doug McDorman
* added section on HTTP status codes for redirects
* added new section on access token privilege restriction based on comments from Johan Peeters

-06

* reworked section 3.8.1
* incorporated Phil Hunt’s feedback
* reworked section on mix-up
* extended section on code leakage via referrer header to also cover state leakage
* added Daniel Fett as author

* replaced text intended to inform WG discussion by recommendations to implementors
* modified example URLs to conform to RFC 2606

-05

* Completed sections on code leakage via referrer header, attacks in browser, mix-up, and CSRF
* Reworked Code Injection Section
* Added reference to OpenID Connect spec
* removed refresh token leakage as respective considerations have been given in section 10.4 of RFC 6749
* first version on open redirection
* incorporated Christian Mainka's review feedback

-04

* Restructured document for better readability
* Added best practices on Token Leakage prevention

-03

* Added section on Access Token Leakage at Resource Server
* incorporated Brian Campbell's findings

-02

* Folded Mix up and Access Token leakage through a bad AS into new section for dynamic OAuth threats
* reworked dynamic OAuth section

-01

* Added references to mitigation methods for token leakage
* Added reference to Token Binding for Authorization Code
* incorporated feedback of Phil Hunt
* fixed numbering issue in attack descriptions in section 2

-00 (WG document)

* turned the ID into a WG document and a BCP
* Added federated app login as topic in Other Topics

Authors' Addresses

Torsten Lodderstedt
SPRIND
Email: torsten@lodderstedt.net

John Bradley
Yubico
Email: ve7jtb@ve7jtb.com

Andrey Labunets
Independent Researcher
Email: isciurus@gmail.com

Daniel Fett
Authlete
Email: mail@danielfett.de

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 12 October 2024

C. Martinez
LACNIC
G. Michaelson
T. Harrison
APNIC
T. Bruijnzeels
RIPE NCC
R. Austein
Dragon Research Labs
10 April 2024

RPKI Signed Object for Trust Anchor Key
draft-ietf-sidrops-signed-tal-15

Abstract

A Trust Anchor Locator (TAL) is used by Relying Parties (RPs) in the Resource Public Key Infrastructure (RPKI) to locate and validate a Trust Anchor (TA) Certification Authority (CA) certificate used in RPKI validation. This document defines an RPKI signed object for a Trust Anchor Key (TAK), that can be used by a TA to signal the location(s) of the accompanying CA certificate for the current key to RPs, as well as the successor key and the location(s) of its CA certificate. This object helps to support planned key rolls without impacting RPKI validation.

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 12 October 2024.

Table of Contents

1. Requirements Notation

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.

2. Overview

A Trust Anchor Locator (TAL) [RFC8630] is used by Relying Parties (RPs) in the Resource Public Key Infrastructure (RPKI) to locate and validate Trust Anchor (TA) Certification Authority (CA) certificates used in RPKI validation. However, until now, there has been no in-band way of notifying RPs of updates to a TAL. In-band notification means that TAs can be more confident of RPs being aware of key roll operations.

This document defines a new RPKI signed object that can be used to document the location(s) of the TA CA certificate for the current TA key, as well as the value of the successor key and the location(s) of its TA CA certificate. This allows RPs to be notified automatically of such changes, and enables TAs to stage a successor key so that planned key rolls can be performed without risking the invalidation of the RPKI tree under the TA. We call this object the Trust Anchor Key (TAK) object.

When RPs are first bootstrapped, they use a TAL to discover the key and location(s) of the CA certificate for a TA. The RP can then retrieve and validate the CA certificate, and subsequently validate the manifest [RFC9286] and Certificate Revocation List (CRL) published by that TA (section 5 of [RFC6487]). However, before processing any other objects, it will first validate the TAK object, if present. If the TAK object lists only the current key, then the RP continues processing as it would in the absence of a TAK object. If the TAK object includes a successor key, the RP starts an acceptance timer, and then continues processing as it would in the absence of a TAK object. If, during the following validation runs up until the expiry of the acceptance timer, the RP has not observed any changes to the keys and certificate URLs listed in the TAK object,

then the RP will fetch the successor key, update its local state with that key and its associated certification location(s), and continue processing using that key.

The primary motivation for this work is being able to migrate from a Hardware Security Module (HSM) produced by one vendor to one produced by another, where the first vendor does not support exporting keys for use by the second. There may be other scenarios in which key rollover is useful, though.

3. TAK Object Definition

The TAK object makes use of the template for RPKI digitally signed objects [RFC6488], which defines a Cryptographic Message Syntax (CMS) [RFC5652] wrapper for the content as well as a generic validation procedure for RPKI signed objects. Therefore, to complete the specification of the TAK object (see Section 4 of [RFC6488]), this document defines:

* The OID (in Section 3.1) that identifies the signed object as being a TAK. (This OID appears within the eContentType in the encapContentInfo object, as well as the content-type signed attribute in the signerInfo object.)

* The ASN.1 syntax for the TAK eContent, in Section 3.2.

* The additional steps required to validate a TAK, in Section 3.3.

3.1. The TAK Object Content Type

This document requests an OID for the TAK object as follows:

```
id-ct-signedTAL OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) ct(1) 50 }
```

This OID MUST appear in both the eContentType in the encapContentInfo object and the content-type signed attribute in the signerInfo object (see [RFC6488]).

3.2. The TAK Object eContent

The content of a TAK object is ASN.1 encoded using the Distinguished Encoding Rules (DER) [X.690], and is defined per the module in Appendix A.

3.2.1. TAKey

This structure defines a TA key, similar to that from [RFC8630]. It contains a sequence of zero or more comments, one or more certificate URIs, and a SubjectPublicKeyInfo.

3.2.1.1. comments

This field is equivalent to the comment section defined in section 2.2 of [RFC8630]. Each comment is human-readable informational UTF-8 text [RFC3629], conforming to the restrictions defined in Section 2 of [RFC5198]. The leading "#" character is omitted.

3.2.1.2. certificateURIs

This field is equivalent to the URI section defined in section 2.2 of [RFC8630]. It MUST contain at least one CertificateURI element. Each CertificateURI element contains the IA5String representation of either an rsync URI [RFC5781], or an HTTPS URI [RFC9110].

3.2.1.3. subjectPublicKeyInfo

This field contains a SubjectPublicKeyInfo (section 4.1.2.7 of [RFC5280]) in DER format [X.690].

3.2.2. TAK

3.2.2.1. version

The version number of the TAK object MUST be 0.

3.2.2.2. current

This field contains the TA key of the repository in which the TAK object is published.

3.2.2.3. predecessor

This field contains the TA key that was in use for this TA immediately prior to the current TA key, if applicable.

3.2.2.4. successor

This field contains the TA key to be used in place of the current key, after expiry of the relevant acceptance timer.

3.3. TAK Object Validation

To determine whether a TAK object is valid, the RP MUST perform the following checks in addition to those specified in [RFC6488]:

* The eContentType OID matches the OID described in Section 3.1.

* The TAK object appears as the product of a TA CA certificate (i.e. the TA CA certificate is itself the issuer of the End-Entity (EE) certificate of the TAK object).

* The TA CA has published only one TAK object in its repository for this key, and this object appears on the manifest as the only entry using the ".tak" extension (see [RFC6481]).

* The EE certificate of this TAK object describes its Internet Number Resources (INRs) using the "inherit" attribute.

* The decoded TAK content conforms to the format defined in Section 3.2.

* The SubjectPublicKeyInfo value of the current TA key in the TAK object matches that of the TA CA certificate used to issue the EE certificate of the TAK object.

If any of these checks does not succeed, the RP MUST ignore the TAK object and proceed as though it were not listed on the manifest.

The RP is not required to compare its current set of certificateURIs for the current key with those listed in the TAK object. The RP MAY alert the user that these sets of certificateURIs do not match, with a view to the user manually updating the set of certificateURIs in their configuration. The RP MUST NOT automatically update its configuration to use these certificateURIs in the event of inconsistency, though, because the migration of users to new certificateURIs should happen by way of the successor key process.

4. TAK Object Generation and Publication

If a TA chooses to use TAK objects to communicate its current, predecessor, and successor keys, then it SHOULD generate and publish TAK objects under each of its keys.

A non-normative guideline for naming this object is that the filename chosen for the TAK object in the publication repository be a value derived from the public key part of the entity's key pair, using the algorithm described for CRLs in section 2.2 of [RFC6481] for generation of filenames. The filename extension of ".tak" MUST be used to denote the object as a TAK.

In order to generate a TAK object, the TA MUST perform the following actions:

* The TA MUST generate a one-time-use EE certificate for the TAK.

* This EE certificate MUST have a unique key pair.

* This EE certificate MUST have a Subject Information Access (SIA) [RFC6487] extension access description field with an accessMethod OID value of id-ad-signedObject, where the associated accessLocation references the publication point of the TAK as an object URL.

* As described in [RFC6487], the EE certificate used for this object must include an [RFC3779] extension. However, because the resource set is irrelevant to this object type, this certificate MUST describe its Internet Number Resources (INRs) using the "inherit" attribute, rather than explicitly describing a resource set.

* This EE certificate MUST have a "notBefore" time that matches or predates the moment that the TAK will be published.

* This EE certificate MUST have a "notAfter" time that reflects the intended duration for which this TAK will be published. If the EE certificate for a TAK object is expired, it MUST no longer be published, but it MAY be replaced by a newly generated TAK object with equivalent content and an updated "notAfter" time.

* The current TA key for the TAK MUST match that of the TA CA certificate under which the TAK was issued.

5. Relying Party Use

Relying Parties MUST keep a record of the current key for each configured TA, as well as the URI(s) where the CA certificate for this key may be retrieved. This record is typically bootstrapped by the use of a pre-configured (and unsigned) TAL file [RFC8630].

When performing top-down validation, RPs MUST first validate and process the TAK object for its current known key, by performing the following steps:

* A CA certificate is retrieved and validated from the known URIs as described in sections 3 and 4 of [RFC8630].

* The manifest and CRL for this certificate are then validated as described in [RFC6487] and [RFC9286].

* The TAK object, if present, is validated as described in Section 3.3.

If the TAK object includes a successor key, then the RP must verify the successor key by doing the following:

* performing top-down validation using the successor key, in order to validate the TAK object for the successor TA;

* ensuring that a valid TAK object exists for the successor TA;

* ensuring that the successor TAK object's current key matches the initial TAK object's successor key; and

* ensuring that the successor TAK object's predecessor key matches the initial TAK object's current key.

If any of these steps fails, then the successor key has failed verification.

If the successor key passes verification, and the RP has not seen that successor key on the previous successful validation run for this TA, then the RP:

* sets an acceptance timer of 30 days for this successor key for this TA;

* cancels the existing acceptance timer for this TA (if applicable); and

* continues standard top-down validation as described in [RFC6487] using the current key.

If the successor key passes verification, and the RP has seen that successor key on the previous successful validation run for this TA:

* if the relevant acceptance timer has not expired, the RP continues standard top-down validation using the current key;

* otherwise, the RP updates its current known key details for this TA to be those of the successor key, and then begins top-down validation again using the successor key.

If the successor key does not pass verification, or if the TAK object does not include a successor key, the RP cancels the existing acceptance timer for this TA (if applicable).

An RP MUST NOT use a successor key for top-down validation outside of the process described above, except for the purpose of testing that the new key is working correctly. This allows a TA to publish a successor key for a period of time, allowing RPs to test it, while still being able to rely on RPs using the current key for their production RPKI operations.

A successor key may have the same SubjectPublicKeyInfo value as the current key: this will be the case where a TA is updating the certificateURIs for that key.

5.1. Manual update of TA key details

A Relying Party may opt not to support the automatic transition of TA key data, as defined in the previous section. An alternative approach is for the Relying Party to alert the user when a new successor key is seen, and also when the relevant acceptance timer has expired. The user can then manually transition to the new TA key data. This process ensures that the benefits of the acceptance timer period are still realised, as compared with TA key update based on a TAL distributed out-of-band by a TA.

6. Maintaining Multiple TA Keys

Although an RP that can process TAK objects will only ever use one key for validation (either the current key, or the successor key, once the relevant acceptance timer has expired), an RP that cannot process TAK objects will continue to use the key details per its TAL (or equivalent manual configuration) indefinitely. As a result, even when a TA is using a TAK object in order to migrate clients to a new key, the TA may have to maintain the previous key for a period of time alongside the new key in order to ensure continuity of service for older clients.

For each TA key that a TA maintains, the signed material for these keys MUST be published under different directories in the context of the 'id-ad-caRepository' and 'id-ad-rpkiManifest' Subject Information Access descriptions contained on the CA certificates [RFC6487]. Publishing objects under the same directory is potentially confusing for RPs, and could lead to object invalidity in the event of file name collisions.

Also, the CA certificates for each maintained key, and the contents published by each key, MUST be equivalent (except for the TAK object). In other words, for the purposes of RPKI validation, it MUST NOT make a difference which of the keys is used as a starting point.

This means that the IP and Autonomous System (AS) resources contained on all current CA certificates for the maintained TA keys MUST be the same. Furthermore, for any delegation of IP and AS resources to a child, the TA MUST have an equivalent CA certificate published under each of its keys. Any updates in delegations MUST be reflected under each of its keys. A TA SHOULD NOT publish any other objects besides a CRL, a Manifest, a single TAK object, and any number of CA certificates for delegation to child CAs.

If a TA uses a single remote publication server for its keys, per [RFC8181], then it MUST include all <publish/> and <withdraw/> Protocol Data Units (PDUs) for the products of each of its keys in a single query, in order to ensure that they will reflect the same content at all times.

If a TA uses multiple publication servers, then the content for different keys will be out of sync at times. The TA SHOULD ensure that the duration of these moments is limited to the shortest possible time. Furthermore, the following should be observed:

* In cases where a CA certificate is revoked, or replaced by a certificate with a reduced set of resources, these changes will not take effect fully until all the relevant repository publication points have been updated. Given that TA key operations are normally performed infrequently, this is unlikely to be a problem: if the revocation or shrinking of an issued CA certificate is staged for days/weeks, then experiencing a delay of several minutes for the repository publication points to be updated is relatively insignificant.

* In cases where a CA certificate is replaced by a certificate with an extended set of resources, the TA MUST inform the receiving CA only after all of its repository publication points have been updated. This ensures that the receiving CA will not issue any products that could be invalid if an RP uses a TA key just before the CA certificate was due to be updated.

Finally, note that the publication locations of CA certificates for delegations to child CAs under each key will be different, and therefore the Authority Information Access 'id-ad-caIssuers' values (section 4.8.7 of [RFC6487]) on certificates issued by the child CAs may not be as expected when performing top-down validation, depending on the TA key that is used. However, these values are not critical to top-down validation, so RPs performing such validation MUST NOT reject a certificate simply because this value is not as expected.

7. Performing TA Key Rolls

In this section we describe how present-day RPKI TAs that use only one key pair, and that do not use TAK objects, can use a TAK object to perform a planned key roll.

7.1. Phase 1: Add a TAK for Key 'A'

Before adding a successor key, a TA may want to confirm that it can maintain a TAK object for its current key only. We will refer to this key as key 'A' throughout this section.

7.2. Phase 2: Add a Key 'B'

The TA can now generate a new key pair for key 'B'. This key MUST now be used to create a new CA certificate for this key, and to issue equivalent CA certificates for delegations to child CAs, as described in Section 6.

At this point, the TA can also construct a new TAL file [RFC8630] for key 'B', and test locally that the validation outcome for the new key is equivalent to that of the other current key(s).

When the TA is certain that both keys are equivalent, and wants to initiate the migration from 'A' to 'B', it issues a new TAK object under key 'A', with key 'A' as the current key for that object, key 'B' as the successor key, and no predecessor key. It also issues a TAK object under key 'B', with key 'B' as the current key for that object, key 'A' as the predecessor key, and no successor key.

Once this has happened, RP clients will start seeing the new key and setting acceptance timers accordingly.

7.3. Phase 3: Update TAL to point to 'B'

At about the time that the TA expects clients to start setting key 'B' as the current key, the TA must release a new TAL file for key 'B'. It SHOULD use a different set of URIs in the TAL compared to the TAK file, so that the TA can learn the proportion of RPs that can successfully validate and use the updated TAK objects.

To support RPs that do not take account of TAK objects, the TA should continue operating key 'A' for a period of time after the expected migration of clients to 'B'. The length of that period of time is a local policy matter for that TA: it might operate the key until no clients are attempting to validate using it, for example.

7.4. Phase 4: Remove Key 'A'

The TA SHOULD now remove all content from the repository used by key 'A', and destroy the private key for key 'A'. RPs attempting to rely on a TAL for key 'A' from this point will not be able to perform RPKI validation for the TA, and will have to update their local state manually, by way of a new TAL file.

8. Using TAK objects to distribute TAL data

Relying Parties must be configured with RPKI Trust Anchor data in order to function correctly. This Trust Anchor data is typically distributed in the Trust Anchor Locator (TAL) format defined in RFC 8630. A TAK object can also serve as a format for distribution of this data, though, because the TAKey data stored in the TAK object contains the same data that would appear in a TAL for the associated Trust Anchor.

Relying Parties may support conversion of TAK objects into TAL files. Relying Parties that support conversion MUST validate the TAK object using the process from section 3.3. One exception to the standard validation process in this context is that a Relying Party MAY treat a TAK object as valid, even though it is associated with a Trust Anchor that the Relying Party is not currently configured to trust. If the Relying Party is relying on this exception when converting a given TAK object, the Relying Party MUST communicate that fact to the user.

When converting a TAK object, a Relying Party MUST default to producing a TAL file based on the 'current' TAKey in the TAK object, though it MAY optionally support producing TAL files based on the 'predecessor' and 'successor' TAKeys.

When converting a TAK object, a Relying Party MUST include in the TAL file any comments from the corresponding TAKey.

If TAK object validation fails, then the Relying Party MUST NOT produce a TAL file based on the TAK object.

Users should be aware that TAK objects distributed out-of-band have similar security properties to TAL files (i.e. there is no authentication). In particular, TAK objects that are not signed by TAs with which the Relying Party is currently configured should only be used if the source that distributes them is one the user trusts to distribute TAL files.

If a Relying Party is not transitioning to new Trust Anchor data using the automatic process described in section 5 or the partially-manual process described in section 5.1, then the user will have to rely on an out-of-band mechanism for validating and updating the Trust Anchor data for the Relying Party. Users in this situation should take similar care when updating a trust anchor using a TAK object file as when using a TAL file to update TA data.

9. Deployment Considerations

9.1. Relying Party Support

Publishing TAK objects while RPs do not support this standard will result in those RPs rejecting these objects. It is not expected that this will result in the invalidation of any other object under a Trust Anchor.

Some RPs may purposefully not support this mechanism: for example, they may be implemented or configured such that they are unable to update local current key data. TAs should take this into consideration when planning key rollover. However, these RPs would ideally still notify their operators of planned key rollovers, so that the operator could update the relevant configuration manually.

9.2. Alternate Transition Models

Alternate models of TAL update exist and are complementary to this mechanism. For example, TAs can liaise directly with RP software developers to include updated and reissued TAL files in new code releases, and use existing code update mechanisms in the RP community to distribute the changes.

Additionally, these non-TA channels for distributing TAL data may themselves rely on monitoring for TAK objects and then updating the TAL data in their distributions or packages accordingly. In this way, TAK objects may be useful even for RPs that don't implement in-band support for the protocol.

Non-TA channels for distributing TAL data should ensure, so far as is possible, that their update mechanisms take account of any changes that a user has made to their local TA key configuration. For example, if a new key is published for a TA, but the non-TA channel's mechanism is able to detect that a user had removed the TA's previous key from their local TA key configuration such that the user no longer relies on it, then the mechanism should not by default add the new key to the user's TA key configuration.

10. Operational Considerations

10.1. Acceptance Timers

Acceptance timers are used in TAK objects in order to permit RPs to test that the new key is working correctly. This in turn means that the TA will be able to gain confidence in the correct functioning of the new key before RPs are relying on that in their production RPKI operations. If a successor key is not working correctly, a TA may remove that key from the current TAK object.

A TA that removes a successor key from a TAK object SHOULD NOT add the same successor key back into the TAK object for that TA. This is because there may be an RP that has fetched the TAK object while the successor key was listed in it, and has started an acceptance timer accordingly, but has not fetched the TAK object during the period when the successor key was not listed in it. If the unchanged successor key is added back into the TA, such an RP will transition to using the new TA key more quickly than other RPs, which may, in turn, make debugging and similar more complicated. A simple way of addressing this problem in a situation where the TA doesn't want to reissue the SubjectPublicKeyInfo content for the successor key that was withdrawn is to update the URL set for the successor key, since RPs must take that URL set into account for the purposes of initiating and cancelling acceptance timers.

11. Security Considerations

11.1. Previous Keys

A TA needs to consider the length of time for which it will maintain previously-current keys and their associated repositories. An RP that is seeded with old TAL data will run for 30 days using the previous key before migrating to the next key, due to the acceptance timer requirements, and this 30-day delay applies to each new key that has been issued since the old TAL data was initially published. It may be better in these instances to have the old publication URLs simply fail to resolve, so that the RP reports an error to its operator and the operator seeds it with up-to-date TAL data immediately.

Once a TA has decided not to maintain a previously-current key and its associated repository, the TA SHOULD destroy that key. The TA SHOULD also reuse the TA CA certificate URLs from the previous TAL data for the next TAL that it generates. These measures will help to mitigate the risk of an adversary gaining access to the key and its associated publication points in order to send invalid/incorrect data to RPs seeded with the TAL data for that key.

11.2. TA Compromise

TAK objects do not offer protection against compromise of the current TA key or the successor TA key. TA key compromise in general is out of scope for this document.

12. IANA Considerations

12.1. Content Type

IANA is asked to register an object identifier for one content type in the "SMI Security for S/MIME CMS Content Type (1.2.840.113549.1.9.16.1)" registry as follows:

Decimal	Description	References
50	id-ct-signedTAL	[section 3.1]

* Description: id-ct-signedTAL
* OID: 1.2.840.113549.1.9.16.1.50
* Specification: [section 3.1]

12.2. Signed Object

IANA is asked to add the following to the "RPKI Signed Objects" registry:

Name	OID	Reference
Trust Anchor Key	1.2.840.113549.1.9.16.1.50	[section 3.1]

IANA is also asked to add the following note to the "RPKI Signed Objects" registry:

> Objects of the types listed in this registry, as well as RPKI resource certificates and CRLs, are expected to be validated using the RPKI.

12.3. File Extension

IANA is asked to add an item for the Signed TAL file extension to the "RPKI Repository Name Scheme" created by [RFC6481] as follows:

Filename Extension	RPKI Object	Reference
.tak	Trust Anchor Key	[this document]

12.4. Module Identifier

IANA is asked to register an object identifier for one module identifier in the "SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0)" registry as follows:

Decimal	Description	References
74	RPKISignedTrustAnchorList-2021	[this document]

* Description: RPKISignedTrustAnchorList-2021
* OID: 1.2.840.113549.1.9.16.0.74
* Specification: [this document]

12.5. Registration of Media Type application/rpki-signed-tal

IANA is asked to register the media type "application/rpki-signed-tal" in the "Media Types" registry as follows:

Type name: application

Subtype name: rpki-signed-tal

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: Carries an RPKI Signed TAL. This media type contains no active content. See the Security Considerations section of RFC XXXX for further information.

Interoperability considerations: N/A

Published specification: RFC XXXX

Applications that use this media type: RPKI operators

Fragment identifier considerations: N/A

Additional information: Content: This media type is for a signed object, as defined in RFC 6488, which contains trust anchor key material as defined in RFC XXXX.

Magic number(s): N/A

File extension(s): .tak

Macintosh file type code(s): N/A

Person & email address to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: sidrops WG

Change controller: IESG

13. Implementation Status

NOTE: Please remove this section and the reference to RFC 7942 prior to publication as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

13.1. APNIC

* Responsible Organization: Asia-Pacific Network Information Centre
* Location: https://github.com/APNIC-net/rpki-signed-tal-demo
* Description: A proof-of-concept for relying party TAK usage.
* Level of Maturity: This is a proof-of-concept implementation.
* Coverage: This implementation includes all of the features described in version 15 of this specification, except for writing TAL files based on TAK data. The repository includes a link to various test TALs that can be used for testing TAK scenarios, too.
* Contact Information: Tom Harrison, tomh@apnic.net

13.2. rpki-client

* Responsible Organization: Job Snijders, the OpenBSD project
* Location: https://www.rpki-client.org
* Description: A relying party implementation which can validate TAKs.

* Level of Maturity: Mature. Trust Anchor operators are encouraged to use rpki-client as part of smoke testing to help ensure high levels of standards compliance when introducing changes, and use rpki-client in a continuous monitoring fashion to help maintain high levels of operational excellence.

* Coverage: This implementation includes all features except TAK acceptance timers.

* Contact information: Job Snijders, job@fastly.com

13.3. rpki-rs

* Responsible Organization: Tim Bruijnzeels, tim@ripe.net

* Location: https://github.com/NLnetLabs/rpki-rs/tree/signed-tal

* Description: Library support for encoding and decoding TAK objects.

* Level of Maturity: This is a proof-of-concept implementation.

* Coverage: This implementation includes support for encoding and decoding TAK objects.

* Contact information: Tim Bruijnzeels, tim@ripe.net

14. Revision History

03 - Last draft under Tim's authorship.

04 - First draft with George's authorship. No substantive revisions.

05 - First draft with Tom's authorship. No substantive revisions.

06 - Rob Kisteleki's critique.

07 - Switch to two-key model.

08 - Keepalive.

09 - Acceptance timers, predecessor keys, no long-lived CRL/MFT.

10 - Using TAK objects for distribution of TAL data.

11 - Manual update guidance, additional security considerations, identifier updates.

12 - TAK object comments.

13 - Removal of compromise text, extra RP support text, key destruction text, media type registration, signed object registry note.

14 - Keepalive.

15 - Additional implementation notes and editorial updates.

15. Acknowledgments

The authors wish to thank Martin Hoffmann for a thorough review of the document, Russ Housley for multiple reviews of the ASN.1 definitions and for providing a new module for the TAK object, Job Snijders for the extensive suggestions around TAK object structure/ distribution and rpki-client implementation work, and Ties de Kock for text/suggestions around TAK/TAL distribution and general security considerations.

16. References

16.1. 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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", RFC 3779, DOI 10.17487/RFC3779, June 2004, <https://www.rfc-editor.org/info/rfc3779>.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/info/rfc5198>.

[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>.

[RFC5781] Weiler, S., Ward, D., and R. Housley, "The rsync URI Scheme", RFC 5781, DOI 10.17487/RFC5781, February 2010, <https://www.rfc-editor.org/info/rfc5781>.

[RFC6481] Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", RFC 6481, DOI 10.17487/RFC6481, February 2012, <https://www.rfc-editor.org/info/rfc6481>.

[RFC6487] Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", RFC 6487, DOI 10.17487/RFC6487, February 2012, <https://www.rfc-editor.org/info/rfc6487>.

[RFC6488] Lepinski, M., Chi, A., and S. Kent, "Signed Object Template for the Resource Public Key Infrastructure (RPKI)", RFC 6488, DOI 10.17487/RFC6488, February 2012, <https://www.rfc-editor.org/info/rfc6488>.

[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>.

[RFC8181] Weiler, S., Sonalker, A., and R. Austein, "A Publication Protocol for the Resource Public Key Infrastructure (RPKI)", RFC 8181, DOI 10.17487/RFC8181, July 2017, <https://www.rfc-editor.org/info/rfc8181>.

[RFC8630] Huston, G., Weiler, S., Michaelson, G., Kent, S., and T. Bruijnzeels, "Resource Public Key Infrastructure (RPKI) Trust Anchor Locator", RFC 8630, DOI 10.17487/RFC8630, August 2019, <https://www.rfc-editor.org/info/rfc8630>.

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

[RFC9286] Austein, R., Huston, G., Kent, S., and M. Lepinski, "Manifests for the Resource Public Key Infrastructure (RPKI)", RFC 9286, DOI 10.17487/RFC9286, June 2022, <https://www.rfc-editor.org/info/rfc9286>.

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

16.2. Informative References

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[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>.

Appendix A. ASN.1 Module

This appendix includes the ASN.1 module for the TAK object.

```
<CODE BEGINS>
RPKISignedTrustAnchorList-2021
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs9(9) smime(16) mod(0) 74 }

DEFINITIONS EXPLICIT TAGS ::=
BEGIN

IMPORTS

CONTENT-TYPE
    FROM CryptographicMessageSyntax-2009 -- in [RFC5911]
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs-9(9) smime(16) modules(0) id-mod-cms-2004-02(41) }

SubjectPublicKeyInfo
    FROM PKIX1Explicit-2009 -- in [RFC5912]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-explicit-02(51) } ;

ct-signedTAL CONTENT-TYPE ::=
    { TYPE TAK IDENTIFIED BY
      id-ct-signedTAL }

id-ct-signedTAL OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) ct(1) 50 }

CertificateURI ::= IA5String

TAKey ::= SEQUENCE {
    comments  SEQUENCE SIZE (0..MAX) OF UTF8String,
    certificateURIs  SEQUENCE SIZE (1..MAX) OF CertificateURI,
    subjectPublicKeyInfo  SubjectPublicKeyInfo
}

TAK ::= SEQUENCE {
    version     INTEGER DEFAULT 0,
    current     TAKey,
    predecessor [0] TAKey OPTIONAL,
    successor   [1] TAKey OPTIONAL
}

END
<CODE ENDS>
```

Authors' Addresses

Carlos Martinez
LACNIC
Rambla Mexico 6125
11400 Montevideo
Uruguay
Email: carlos@lacnic.net
URI: https://www.lacnic.net/

George G. Michaelson
Asia Pacific Network Information Centre
6 Cordelia St
South Brisbane QLD 4101
Australia
Email: ggm@apnic.net

Tom Harrison
Asia Pacific Network Information Centre
6 Cordelia St
South Brisbane QLD 4101
Australia
Email: tomh@apnic.net

Tim Bruijnzeels
RIPE NCC
Stationsplein 11
Amsterdam
Netherlands
Email: tim@ripe.net
URI: https://www.ripe.net/

Rob Austein
Dragon Research Labs
Email: sra@hactrn.net

TICTOC Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 October 2024

D.A. Arnold
Meinberg-USA
H.G. Gerstung
Meinberg

3 April 2024

Enterprise Profile for the Precision Time Protocol With Mixed Multicast and Unicast messages
draft-ietf-tictoc-ptp-enterprise-profile-26

Abstract

This document describes a PTP Profile for the use of the Precision Time Protocol in an IPv4 or IPv6 Enterprise information system environment. The PTP Profile uses the End-to-End delay measurement mechanism, allows both multicast and unicast Delay Request and Delay Response messages.

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 5 October 2024.

Table of Contents

1. Introduction

The Precision Time Protocol ("PTP"), standardized in IEEE 1588, has been designed in its first version (IEEE 1588-2002) with the goal to minimize configuration on the participating nodes. Network communication was based solely on multicast messages, which unlike NTP did not require that a receiving node in IEEE 1588-2019 [IEEE1588] need to know the identity of the time sources in the network. This document describes clock roles and PTP Port states using the optional alternative terms timeTransmitter, in stead of master, and timeReceiver, in stead of slave, as defined in the IEEE 1588g [IEEE1588g] amendment to IEEE 1588-2019 [IEEE1588] .

The "Best TimeTransmitter Clock Algorithm" (IEEE 1588-2019 [IEEE1588] Subclause 9.3), a mechanism that all participating PTP nodes MUST follow, set up strict rules for all members of a PTP domain to determine which node MUST be the active reference time source (Grandmaster). Although the multicast communication model has advantages in smaller networks, it complicated the application of PTP in larger networks, for example in environments like IP based telecommunication networks or financial data centers. It is considered inefficient that, even if the content of a message applies only to one receiver, it is forwarded by the underlying network (IP) to all nodes, requiring them to spend network bandwidth and other resources, such as CPU cycles, to drop the message.

The third edition of the standard (IEEE 1588-2019) defines PTPv2.1 and includes the possibility to use unicast communication between the PTP nodes in order to overcome the limitation of using multicast messages for the bi-directional information exchange between PTP nodes. The unicast approach avoided that. In PTP domains with a lot of nodes, devices had to throw away more than 99% of the received multicast messages because they carried information for some other node.

PTPv2.1 also includes PTP Profiles (IEEE 1588-2019 [IEEE1588] subclause 20.3). This construct allows organizations to specify selections of attribute values and optional features, simplifying the configuration of PTP nodes for a specific application. Instead of having to go through all possible parameters and configuration options and individually set them up, selecting a PTP Profile on a PTP node will set all the parameters that are specified in the PTP Profile to a defined value. If a PTP Profile definition allows multiple values for a parameter, selection of the PTP Profile will set the profile-specific default value for this parameter. Parameters not allowing multiple values are set to the value defined in the PTP Profile. Many PTP features and functions are optional, and a PTP Profile should also define which optional features of PTP are required, permitted, and prohibited. It is possible to extend the PTP standard with a PTP Profile by using the TLV mechanism of PTP (see IEEE 1588-2019 [IEEE1588] subclause 13.4), defining an optional Best TimeTransmitter Clock Algorithm and a few other ways. PTP has its own management protocol (defined in IEEE 1588-2019 [IEEE1588] subclause 15.2) but allows a PTP Profile to specify an alternative management mechanism, for example NETCONF.

In this document the term PTP Port refers to a logical access point of a PTP instantiation for PTP communincation in a network.

2. 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 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Technical Terms

* Acceptable TimeTransmitter Table: A PTP timeReceiver Clock may maintain a list of timeTransmitters which it is willing to synchronize to.

* Alternate timeTransmitter: A PTP timeTransmitter Clock, which is not the Best timeTransmitter, may act as a timeTransmitter with the Alternate timeTransmitter flag set on the messages it sends.

* Announce message: Contains the timeTransmitter Clock properties of a timeTransmitter Clock. Used to determine the Best TimeTransmitter.

* Best timeTransmitter: A clock with a PTP Port in the timeTransmitter state, operating as the Grandmaster of a PTP domain.

* Best TimeTransmitter Clock Algorithm: A method for determining which state a PTP Port of a PTP clock should be in. The state decisions lead to the formation of a clock spanning tree for a PTP domain.

* Boundary Clock: A device with more than one PTP Port. Generally Boundary Clocks will have one PTP Port in timeReceiver state to receive timing and other PTP Ports in timeTransmitter state to re-distribute the timing.

* Clock Identity: In IEEE 1588-2019 this is a 64-bit number assigned to each PTP clock which MUST be globally unique. Often it is derived from the Ethernet MAC address.

* Domain: Every PTP message contains a domain number. Domains are treated as separate PTP systems in the network. Clocks, however, can combine the timing information derived from multiple domains.

* End-to-End delay measurement mechanism: A network delay measurement mechanism in PTP facilitated by an exchange of messages between a timeTransmitter Clock and a timeReceiver Clock. These messages might traverse Transparent Clocks and PTP unaware switches. This mechanism might not work properly if the Sync and Delay Request messages traverse different network paths.

* Grandmaster: the primary timeTransmitter Clock within a domain of a PTP system

* IEEE 1588: The timing and synchronization standard which defines PTP, and describes the node, system, and communication properties necessary to support PTP.

* TimeTransmitter Clock: a clock with at least one PTP Port in the timeTransmitter state.

* NTP: Network Time Protocol, defined by RFC 5905, see RFC 5905 [RFC5905]

* Ordinary Clock: A clock that has a single Precision Time Protocol PTP Port in a domain and maintains the timescale used in the domain. It may serve as a timeTransmitter Clock, or be a timeReceiver Clock.

* Peer-to-Peer delay measurement mechanism: A network delay measurement mechanism in PTP facilitated by an exchange of messages over the link between adjacent devices in a network. This mechanism might not work properly unless all devices in the network support PTP and the Peer-to-peer measurement mechanism.

* Preferred timeTransmitter: A device intended to act primarily as the Grandmaster of a PTP system, or as a back up to a Grandmaster.

* PTP: The Precision Time Protocol: The timing and synchronization protocol defined by IEEE 1588.

* PTP Port: An interface of a PTP clock with the network. Note that there may be multiple PTP Ports running on one physical interface, for example, mulitple unicast timeReceivers which talk to several Grandmaster Clocks in different PTP Domains.

* PTP Profile: A set of constraints on the options and features of PTP, designed to optimize PTP for a specific use case or industry. The profile specifies what is required, allowed and forbidden among options and attribute values of PTP.

* PTPv2.1: Refers specifically to the version of PTP defined by IEEE 1588-2019.

* Rogue timeTransmitter: A clock with a PTP Port in the timeTransmitter state, even though it should not be in the timeTransmitter state according to the Best TimeTransmitter Clock Algorithm, and does not set the Alternate timeTransmitter flag.

* TimeReceiver Clock: a clock with at least one PTP Port in the timeReceiver state, and no PTP Ports in the timeTransmitter state.

* TimeReceiver Only clock: An Ordinary Clock which cannot become a timeTransmitter Clock.

* TLV: Type Length Value, a mechanism for extending messages in networked communications.

* Transparent Clock. A device that measures the time taken for a PTP event message to transit the device and then updates the message with a correction for this transit time.

* Unicast Discovery: A mechanism for PTP timeReceivers to establish a unicast communication with PTP timeTransmitters using a configured table of timeTransmitter IP addresses and Unicast Message Negotiation.

* Unicast Negotiation: A mechanism in PTP for timeReceiver Clocks to negotiate unicast Sync, Announce and Delay Request message transmission rates from timeTransmitters.

4. Problem Statement

This document describes a version of PTP intended to work in large enterprise networks. Such networks are deployed, for example, in financial corporations. It is becoming increasingly common in such networks to perform distributed time tagged measurements, such as one-way packet latencies and cumulative delays on software systems spread across multiple computers. Furthermore, there is often a desire to check the age of information time tagged by a different machine. To perform these measurements, it is necessary to deliver a common precise time to multiple devices on a network. Accuracy currently required in the Financial Industry range from 100 microseconds to 1 nanoseconds to the Grandmaster. This PTP Profile does not specify timing performance requirements, but such requirements explain why the needs cannot always be met by NTP, as commonly implemented. Such accuracy cannot usually be achieved with a traditional time transfer such as NTP, without adding non-standard customizations such as on-path support, similar to what is done in

PTP with Transparent Clocks and Boundary Clocks. Such PTP support is commonly available in switches and routers, and many such devices have already been deployed in networks. Because PTP has a complex range of features and options it is necessary to create a PTP Profile for enterprise networks to achieve interoperability between equipment manufactured by different vendors.

Although enterprise networks can be large, it is becoming increasingly common to deploy multicast protocols, even across multiple subnets. For this reason, it is desired to make use of multicast whenever the information going to many destinations is the same. It is also advantageous to send information which is only relevant to one device as a unicast message. The latter can be essential as the number of PTP timeReceivers becomes hundreds or thousands.

PTP devices operating in these networks need to be robust. This includes the ability to ignore PTP messages which can be identified as improper, and to have redundant sources of time.

Interoperability among independent implementations of this PTP Profile has been demonstrated at the ISPCS Plugfest ISPCS [ISPCS].

5. Network Technology

This PTP Profile MUST operate only in networks characterized by UDP RFC 768 [RFC0768] over either IPv4 RFC 791 [RFC0791] or IPv6 RFC 8200 [RFC8200], as described by Annexes C and D in IEEE 1588 [IEEE1588] respectively. A network node MAY include multiple PTP instances running simultaneously. IPv4 and IPv6 instances in the same network node MUST operate in different PTP Domains. PTP Clocks which communicate using IPv4 can transfer time to PTP Clocks using IPv6, or the reverse, if and only if, there is a network node which simultaneously communicates with both PTP domains in the different IP versions.

The PTP system MAY include switches and routers. These devices MAY be Transparent Clocks, Boundary Clocks, or neither, in any combination. PTP Clocks MAY be Preferred timeTransmitters, Ordinary Clocks, or Boundary Clocks. The Ordinary Clocks may be TimeReceiver Only Clocks, or be timeTransmitter capable.

Note that clocks SHOULD always be identified by their Clock ID and not the IP or Layer 2 address. This is important since Transparent Clocks will treat PTP messages that are altered at the PTP application layer as new IP packets and new Layer 2 frames when the PTP messages are retranmitted. In IPv4 networks some clocks might be hidden behind a NAT, which hides their IP addresses from the rest of

the network. Note also that the use of NATs may place limitations on the topology of PTP networks, depending on the port forwarding scheme employed. Details of implementing PTP with NATs are out of scope of this document.

PTP, similar to NTP, assumes that the one-way network delay for Sync messages and Delay Response messages are the same. When this is not true it can cause errors in the transfer of time from the timeTransmitter to the timeReceiver. It is up to the system integrator to design the network so that such effects do not prevent the PTP system from meeting the timing requirements. The details of network asymmetry are outside the scope of this document. See for example, ITU-T G.8271 [G8271].

6. Time Transfer and Delay Measurement

TimeTransmitter Clocks, Transparent Clocks and Boundary Clocks MAY be either one-step clocks or two-step clocks. TimeReceiver Clocks MUST support both behaviors. The End-to-End Delay measurement method MUST be used.

Note that, in IP networks, Sync messages and Delay Request messages exchanged between a timeTransmitter and timeReceiver do not necessarily traverse the same physical path. Thus, wherever possible, the network SHOULD be engineered so that the forward and reverse routes traverse the same physical path. Traffic engineering techniques for path consistency are out of scope of this document.

Sync messages MUST be sent as PTP event multicast messages (UDP port 319) to the PTP primary IP address. Two step clocks MUST send Follow-up messages as PTP general multicast messages (UDP port 320). Announce messages MUST be sent as multicast messages (UDP port 320) to the PTP primary address. The PTP primary IP address is 224.0.1.129 for IPv4 and FF0X:0:0:0:0:0:0:181 for IPv6, where X can be a value between 0x0 and 0xF, see IEEE 1588 [IEEE1588] Annex D, Section D.3. These addresses are aloted by IANA, see the Ipv6 Multicast Address Space Registry [IPv6Registry]

Delay Request messages MAY be sent as either multicast or unicast PTP event messages. TimeTransmitter Clocks MUST respond to multicast Delay Request messages with multicast Delay Response PTP general messages. TimeTransmitter Clocks MUST respond to unicast Delay Request PTP event messages with unicast Delay Response PTP general messages. This allows for the use of Ordinary Clocks which do not support the Enterprise Profile, if they are timeReceiver Only Clocks.

Clocks SHOULD include support for multiple domains. The purpose is to support multiple simultaneous timeTransmitters for redundancy. Leaf devices (non-forwarding devices) can use timing information from multiple timeTransmitters by combining information from multiple instantiations of a PTP stack, each operating in a different PTP Domain. Redundant sources of timing can be ensembled, and/or compared to check for faulty timeTransmitter Clocks. The use of multiple simultaneous timeTransmitters will help mitigate faulty timeTransmitters reporting as healthy, network delay asymmetry, and security problems. Security problems include on-path attacks such as delay attacks, packet interception / manipulation attacks. Assuming the path to each timeTransmitter is different, failures malicious or otherwise would have to happen at more than one path simultaneously. Whenever feasible, the underlying network transport technology SHOULD be configured so that timing messages in different domains traverse different network paths.

7. Default Message Rates

The Sync, Announce, and Delay Request default message rates MUST each be once per second. The Sync and Delay Request message rates MAY be set to other values, but not less than once every 128 seconds, and not more than 128 messages per second. The Announce message rate MUST NOT be changed from the default value. The Announce Receipt Timeout Interval MUST be three Announce Intervals for Preferred TimeTransmitters, and four Announce Intervals for all other timeTransmitters.

The logMessageInterval carried in the unicast Delay Response message MAY be set to correspond to the timeTransmitter ports preferred message period, rather than 7F, which indicates message periods are to be negotiated. Note that negotiated message periods are not allowed, see forbidden PTP options (Section 13).

8. Requirements for TimeTransmitter Clocks

TimeTransmitter Clocks MUST obey the standard Best TimeTransmitter Clock Algorithm from IEEE 1588 [IEEE1588]. PTP systems using this PTP Profile MAY support multiple simultaneous Grandmasters if each active Grandmaster is operating in a different PTP domain.

A PTP Port of a clock MUST NOT be in the timeTransmitter state unless the clock has a current value for the number of UTC leap seconds.

If a unicast negotiation signaling message is received it MUST be ignored.

In PTP Networks that contain Transparent Clocks, timeTransmitters might receive Delay Request messages that no longer contains the IP Addresses of the timeReceivers. This is because Transparent Clocks might replace the IP address of Delay Requests with their own IP address after updating the Correction Fields. For this deployment scenario timeTransmitters will need to have configured tables of timeReceivers' IP addresses and associated Clock Identities in order to send Delay Responses to the correct PTP Nodes.

9. Requirements for TimeReceiver Clocks

TimeReceiver Clocks MUST be able to operate properly in a network which contains multiple timeTransmitters in multiple domains. TimeReceivers SHOULD make use of information from all the timeTransmitters in their clock control subsystems. TimeReceiver Clocks MUST be able to operate properly in the presence of a rogue timeTransmitter. TimeReceivers SHOULD NOT Synchronize to a timeTransmitter which is not the Best TimeTransmitter in its domain. TimeReceivers will continue to recognize a Best TimeTransmitter for the duration of the Announce Time Out Interval. TimeReceivers MAY use an Acceptable TimeTransmitter Table. If a timeTransmitter is not an Acceptable timeTransmitter, then the timeReceiver MUST NOT synchronize to it. Note that IEEE 1588-2019 requires timeReceiver Clocks to support both two-step or one-step timeTransmitter Clocks. See IEEE 1588 [IEEE1588], subClause 11.2.

Since Announce messages are sent as multicast messages timeReceivers can obtain the IP addresses of a timeTransmitter from the Announce messages. Note that the IP source addresses of Sync and Follow-up messages might have been replaced by the source addresses of a Transparent Clock, so, timeReceivers MUST send Delay Request messages to the IP address in the Announce message. Sync and Follow-up messages can be correlated with the Announce message using the Clock ID, which is never altered by Transparent Clocks in this PTP Profile.

10. Requirements for Transparent Clocks

Transparent Clocks MUST NOT change the transmission mode of an Enterprise Profile PTP message. For example, a Transparent Clock MUST NOT change a unicast message to a multicast message. Transparent Clocks SHOULD support multiple domains. Transparent Clocks which syntonize to the timeTransmitter Clock might need to maintain separate clock rate offsets for each of the supported domains.

11. Requirements for Boundary Clocks

Boundary Clocks SHOULD support multiple simultaneous PTP domains. This will require them to maintain separate clocks for each of the domains supported, at least in software. Boundary Clocks MUST NOT combine timing information from different domains.

12. Management and Signaling Messages

PTP Management messages MAY be used. Management messages intended for a specific clock, i.e. the IEEE 1588 [IEEE1588] defined attribute targetPortIdentity.clockIdentity is not set to All 1s, MUST be sent as a unicast message. Similarly, if any signaling messages are used they MUST also be sent as unicast messages whenever the message is intended soley for a specific PTP Node.

13. Forbidden PTP Options

Clocks operating in the Enterprise Profile MUST NOT use Peer-to-Peer timing for delay measurement. Grandmaster Clusters are NOT ALLOWED. The Alternate TimeTransmitter option is also NOT ALLOWED. Clocks operating in the Enterprise Profile MUST NOT use Alternate Timescales. Unicast discovery and unicast negotiation MUST NOT be used. Clocks operating in the Enterprise Profile MUST NOT use any optional feature that requires Announce messages to be altered by Transparent Clocks, as this would require the Transparent Clock to change the source address and prevent the timeReceiver nodes from discovering the protocol address of the timeTransmitter.

14. Interoperation with IEEE 1588 Default Profile

Clocks operating in the Enterprise Profile will interoperate with clocks operating in the Default Profile described in IEEE 1588 [IEEE1588] Annex I.3. This variant of the Default Profile uses the End-to-End delay measurement mechanism. In addition, the Default Profile would have to operate over IPv4 or IPv6 networks, and use management messages in unicast when those messages are directed at a specific clock. If either of these requirements are not met than Enterprise Profile clocks will not interoperate with Annex I.3 Default Profile Clocks. The Enterprise Profile will not interoperate with the Annex I.4 variant of the Default Profile which requires use of the Peer-to-Peer delay measurement mechanism.

Enterprise Profile Clocks will interoperate with clocks operating in other PTP Profiles if the clocks in the other PTP Profiles obey the rules of the Enterprise Profile. These rules MUST NOT be changed to achieve interoperability with other PTP Profiles.

15. Profile Identification

The IEEE 1588 standard requires that all PTP Profiles provide the following identifying information.

PTP Profile:
Enterprise Profile
Version: 1.0
Profile identifier: 00-00-5E-00-01-00

This PTP Profile was specified by the IETF

A copy may be obtained at
https://datatracker.ietf.org/wg/tictoc/documents

16. Acknowledgements

The authors would like to thank Richard Cochran, Kevin Gross, John Fletcher, Laurent Montini and many other members of IETF for reviewing and providing feedback on this draft.

This document was initially prepared using 2-Word-v2.0.template.dot and has later been converted manually into xml format using an xml2rfc template.

17. IANA Considerations

There are no IANA requirements in this specification.

18. Security Considerations

Protocols used to transfer time, such as PTP and NTP can be important to security mechanisms which use time windows for keys and authorization. Passing time through the networks poses a security risk since time can potentially be manipulated. The use of multiple simultaneous timeTransmitters, using multiple PTP domains can mitigate problems from rogue timeTransmitters and on-path attacks. Note that Transparent Clocks alter PTP content on-path, but in a manner specified in IEEE 1588-2019 [IEEE1588] that helps with time transfer accuracy. See sections 9 and 10. Additional security mechanisms are outside the scope of this document.

PTP native management messages SHOULD NOT be used, due to the lack of a security mechanism for this option. Secure management can be obtained using standard management mechanisms which include security, for example NETCONF NETCONF [RFC6241].

General security considerations of time protocols are discussed in RFC 7384 [RFC7384].

19. References

19.1. Normative References

[IEEE1588] Institute of Electrical and Electronics Engineers, "IEEE std. 1588-2019, "IEEE Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems."", November 2019, <https://www.ieee.org>.

[IEEE1588g] Institute of Electrical and Electronics Engineers, "IEEE std. 1588g-2022, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems Amendment 2: Master-Slave Optional Alternative Terminology"", December 2022, <https://www.ieee.org>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 2119, DOI 10.17487/RFC2119, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

19.2. Informative References

[G8271] International Telecommunication Union, "ITU-T G.8271/ Y.1366, "Time and Phase Synchronization Aspects of Packet Networks"", March 2020, <https://www.itu.int>.

[IPv6Registry]
Venaas, S., "IPv6 Multicast Address Space Registry", February 2024, <https://iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml>.

[ISPCS] Arnold, D., "Plugfest Report", October 2017, <https://www.ispcs.org>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

Authors' Addresses

Doug Arnold
Meinberg-USA
3 Concord Rd
Shrewsbury, Massachusetts 01545
United States of America
Email: doug.arnold@meinberg-usa.com

Heiko Gerstung
Meinberg
Lange Wand 9
31812 Bad Pyrmont
Germany
Email: heiko.gerstung@meinberg.de

Transport Layer Security
Internet-Draft
Intended status: Informational
Expires: 1 November 2024

M. Thomson
Mozilla

30 April 2024

The SSLKEYLOGFILE Format for TLS
draft-ietf-tls-keylogfile-02

Abstract

A format that supports the logging information about the secrets used in a TLS connection is described. Recording secrets to a file in SSLKEYLOGFILE format allows diagnostic and logging tools that use this file to decrypt messages exchanged by TLS endpoints.

About This Document

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

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

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

Source for this draft and an issue tracker can be found at https://github.com/tlswg/sslkeylogfile.

Status of This Memo

This Internet-Draft will expire on 1 November 2024.

Table of Contents

1. Introduction

Debugging or analyzing protocols can be challenging when TLS [TLS13] is used to protect the content of communications. Inspecting the content of encrypted messages in diagnostic tools can enable more thorough analysis.

Over time, multiple TLS implementations have informally adopted a file format that logging the secret values generated by the TLS key schedule. In many implementations, the file that the secrets are logged to is specified in an environment variable named "SSLKEYLOGFILE", hence the name of SSLKEYLOGFILE format. Note the use of "SSL" as this convention originally predates the adoption of TLS as the name of the protocol.

This document describes the SSLKEYLOGFILE format. This format can be used for TLS 1.2 [TLS12] and TLS 1.3 [TLS13]. The format also supports earlier TLS versions, though use of earlier versions is forbidden [RFC8996]. This format can also be used with DTLS [DTLS13], QUIC [RFC9000][RFC9001], and other protocols that also use the TLS key schedule. Use of this format could complement other protocol-specific logging such as QLOG [QLOG].

1.1. Applicability Statement

The artifact that this document describes - if made available to entities other than endpoints - completely undermines the core guarantees that TLS provides. This format is intended for use in systems where TLS only protects test data. While the access that this information provides to TLS connections can be useful for diagnosing problems while developing systems, this mechanism MUST NOT be used in a production system.

1.2. Conventions and Definitions

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

2. The SSLKEYLOGFILE Format

A file in SSLKEYLOGFILE format is a text file. This document specifies the character encoding as UTF-8 [RFC3629]. Though the format itself only includes ASCII characters [RFC0020], comments MAY contain other characters. Though Unicode is permitted in comments, the file MUST NOT contain a Unicode byte order mark (U+FEFF).

Lines are terminated using the line ending convention of the platform on which the file is generated. Tools that process these files MUST accept CRLF (U+13 followed by U+10), CR (U+13), or LF (U+10) as a line terminator. Lines are ignored if they are empty or if the first character is an octothorp character ('#', U+23). Other lines of the file each contain a single secret.

Implementations that record secrets to a file do so continuously as those secrets are generated.

Each secret is described using a single line composed of three values that are separated by a single space character (U+20). These values are:

label: The label identifies the type of secret that is being conveyed; see Section 2.1 for a description of the labels that are defined in this document.

client_random: The 32-byte value of the Random field from the ClientHello message that established the TLS connection. This value is encoded as 64 hexadecimal characters. Including this field allows a single file to include secrets from multiple connections and for the secrets to be applied to the correct connection, though this depends on being able to match records to the correct ClientHello message.

secret: The value of the identified secret for the identified connection. This value is encoded in hexadecimal, with a length that depends on the size of the secret.

For the hexadecimal values of the client_random or secret, no convention exists for the case of characters 'a' through 'f' (or 'A' through 'F'). Files can be generated with either, so either form MUST be accepted when processing a file.

Diagnostic tools that accept files in this format might choose to ignore lines that do not conform to this format in the interest of ensuring that secrets can be obtained from corrupted files.

Logged secret values are not annotated with the cipher suite or other connection parameters. A record of the TLS handshake might therefore be needed to use the logged secrets.

2.1. Secret Labels for TLS 1.3

An implementation of TLS 1.3 produces a number of values as part of the key schedule (see Section 7.1 of [TLS13]). Each of the following labels correspond to the equivalent secret produced by the key schedule:

CLIENT_EARLY_TRAFFIC_SECRET:
This secret is used to protect records sent by the client as early data, if early data is attempted by the client. Note that a server that rejects early data will not log this secret, though a client that attempts early data can do so unconditionally.

EARLY_EXPORTER_MASTER_SECRET:
This secret is used for early exporters. Like the CLIENT_EARLY_TRAFFIC_SECRET, this is only generated when early data is attempted and might not be logged by a server if early data is rejected.

CLIENT_HANDSHAKE_TRAFFIC_SECRET:
: This secret is used to protect handshake records sent by the client.

SERVER_HANDSHAKE_TRAFFIC_SECRET:
: This secret is used to protect handshake records sent by the server.

CLIENT_TRAFFIC_SECRET_0:
: This secret is used to protect application_data records sent by the client immediately after the handshake completes. This secret is identified as client_application_traffic_secret_0 in the TLS 1.3 key schedule.

SERVER_TRAFFIC_SECRET_0:
: This secret is used to protect application_data records sent by the server immediately after the handshake completes. This secret is identified as server_application_traffic_secret_0 in the TLS 1.3 key schedule.

EXPORTER_SECRET:
: This secret is used in generating exporters Section 7.5 of [TLS13].

These labels all appear in uppercase in the key log, but they correspond to lowercase labels in the TLS key schedule (Section 7.1 of [TLS13]), except for the application data secrets as noted. For example, "EXPORTER_SECRET" in the log file corresponds to the secret named exporter_secret.

Note that the order that labels appear here corresponds to the order in which they are presented in [TLS13], but there is no guarantee that implementations will log secrets strictly in this order.

Key updates (Section 7.2 of [TLS13]) result in new secrets being generated for protecting application_data records. The label used for these secrets comprises a base label of "CLIENT_TRAFFIC_SECRET_" for a client or "SERVER_TRAFFIC_SECRET_" for a server, plus the decimal value of a counter. This counter identifies the number of key updates that occurred to produce this secret. This counter starts at 0, which produces the first application data traffic secret, as above. Note that with knowledge of "_TRAFFIC_SECRET_N", all subsequent application data traffic secret can be derived without any additional information.

2.2. Secret Labels for TLS 1.2

An implementation of TLS 1.2 [TLS12] (and also earlier versions) use the label "CLIENT_RANDOM" to identify the "master" secret for the connection.

3. Security Considerations

Access to the content of a file in SSLKEYLOGFILE format allows an attacker to break the confidentiality and integrity protection on any TLS connections that are included in the file. This includes both active connections and connections for which encrypted records were previously stored. Ensuring adequate access control on these files therefore becomes very important.

Implementations that support logging this data need to ensure that logging can only be enabled by those who are authorized. Allowing logging to be initiated by any entity that is not otherwise authorized to observe or modify the content of connections for which secrets are logged could represent a privilege escalation attack. Implementations that enable logging also need to ensure that access to logged secrets is limited, using appropriate file permissions or equivalent access control mechanisms.

In order to support decryption, the secrets necessary to remove record protection are logged. However, as the keys that can be derived from these secrets are symmetric, an adversary with access to these secrets is also able to encrypt data for an active connection. This might allow for injection or modification of application data on a connection that would otherwise be protected by TLS.

As some protocols rely on TLS for generating encryption keys, the SSLKEYLOGFILE format includes keys that identify the secret used in TLS exporters or early exporters (Section 7.5 of [TLS13]. Knowledge of these secrets can enable more than inspection or modification of encrypted data, depending on how an application protocol uses exporters. For instance, exporters might be used for session bindings (e.g., [RFC8471]), authentication (e.g., [RFC9261]), or other derived secrets that are used in application context. An adversary that obtains these secrets might be able to use this information to attack these applications. A TLS implementation might either choose to omit these secrets in contexts where the information might be abused or require separate authorization to enable logging of exporter secrets.

Using an environment variable, such as SSLKEYLOGFILE, to enable logging implies that access to the launch context for the application is needed to authorize logging. On systems that support specially-

named files, logs might be directed to these names so that logging does not result in storage, but enable consumption by other programs. In both cases, applications might require special authorization or they might rely on system-level access control to limit access to these capabilities.

Forward secrecy guarantees provided in TLS 1.3 (see Section 1.2 and Appendix E.1 of [RFC8446]) and some modes of TLS 1.2 (such as those in Sections 2.2 and 2.4 of [RFC4492]) do not hold if key material is recorded. Access to key material allows an attacker to decrypt data exchanged in any previously logged TLS connections.

Logging the TLS 1.2 "master" secret provides the recipient of that secret far greater access to an active connection than TLS 1.3 secrets. In addition to reading and altering protected messages, the TLS 1.2 "master" secret confers the ability to resume the connection and impersonate either endpoint, insert records that result in renegotiation, and forge Finished messages. Implementations can avoid the risks associated with these capabilities by not logging this secret value.

4. IANA Considerations

The "application/sslkeylogfile" media type can be used to describe content in the SSLKEYLOGFILE format. IANA [has added/is requested to add] the following information to the "Media Types" registry at https://www.iana.org/assignments/media-types (https://www.iana.org/assignments/media-types):

Type name: application
Subtype name: sslkeylogfile
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: UTF-8 without BOM, or ASCII only
Security considerations: See Section 3.
Interoperability considerations: Line endings might differ from platform convention
Published specification: RFC XXXX (RFC Editor: please update)
Applications that use this media type: Diagnostic and analysis tools that need to decrypt data that is otherwise protected by TLS.
Fragment identifier considerations: N/A
Additional information: Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): N/A
Macintosh file type code(s): N/A
Person & email address to contact for further information: TLS WG (tls@ietf.org)
Intended usage: COMMON

Restrictions on usage: N/A
Author: IETF TLS Working Group
Change controller: IESG

5. References

5.1. Normative References

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

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

[TLS12] 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/rfc/rfc5246>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-10, 3 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-10>.

5.2. Informative References

[DTLS13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[QLOG] Marx, R., Niccolini, L., Seemann, M., and L. Pardue, "Main logging schema for qlog", Work in Progress, Internet-Draft, draft-ietf-quic-qlog-main-schema-08, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-main-schema-08>.

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/rfc/rfc20>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <https://www.rfc-editor.org/rfc/rfc4492>.

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

[RFC8471] Popov, A., Ed., Nystroem, M., Balfanz, D., and J. Hodges, "The Token Binding Protocol Version 1.0", RFC 8471, DOI 10.17487/RFC8471, October 2018, <https://www.rfc-editor.org/rfc/rfc8471>.

[RFC8996] Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <https://www.rfc-editor.org/rfc/rfc8996>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[RFC9261] Sullivan, N., "Exported Authenticators in TLS", RFC 9261, DOI 10.17487/RFC9261, July 2022, <https://www.rfc-editor.org/rfc/rfc9261>.

Appendix A. Example

The following is a sample of a file in this format, including secrets from two TLS 1.3 connections.

```
# NOTE: '\' line wrapping per RFC 8792

CLIENT_HANDSHAKE_TRAFFIC_SECRET \
  cf34899b3dcb8c9fe7160ceaf95d354a294793b67a2e49cb9cca4d69b43593a0 \
  be4a28d81ce41242ff31c6d8a6615852178f2cd75eaca2ee8768f9ed51282b38
SERVER_HANDSHAKE_TRAFFIC_SECRET \
  cf34899b3dcb8c9fe7160ceaf95d354a294793b67a2e49cb9cca4d69b43593a0 \
  258179721fa704e2f1ee16688b4b0419967ddea5624cd5ad0863288dc5ead35f
CLIENT_HANDSHAKE_TRAFFIC_SECRET \
  b2eb93b8ddab8c228993567947bca1e133736980c22754687874e3896f7d6d0a \
  59ec0981b211a743f22d5a46a1fc77a2b230e16ef0de6d4e418abfe90eff10bf
SERVER_HANDSHAKE_TRAFFIC_SECRET \
  b2eb93b8ddab8c228993567947bca1e133736980c22754687874e3896f7d6d0a \
  a37fe4d3b6c9a6a372396b1562f6f8a40c1c3f85f1aa9b02d5ed46c4a1301365
CLIENT_TRAFFIC_SECRET_0 \
  cf34899b3dcb8c9fe7160ceaf95d354a294793b67a2e49cb9cca4d69b43593a0 \
  e9ca165bcb762fab8086068929d26c532e90ef2e2daa762d8b52346951a34c02
SERVER_TRAFFIC_SECRET_0 \
  cf34899b3dcb8c9fe7160ceaf95d354a294793b67a2e49cb9cca4d69b43593a0 \
  4f93c61ac1393008d4c820f3723db3c67494f06574b65fcc21c9eef22f90071a
EXPORTER_SECRET \
  cf34899b3dcb8c9fe7160ceaf95d354a294793b67a2e49cb9cca4d69b43593a0 \
  011c900833468f837f7c55d836b2719beebd39b1648fdeda58772f48d94a1ffa
CLIENT_TRAFFIC_SECRET_0 \
  b2eb93b8ddab8c228993567947bca1e133736980c22754687874e3896f7d6d0a \
  e9160bca1a531d871f5ecf51943d8cfb88833adeccf97701546b5fb93e030d79
SERVER_TRAFFIC_SECRET_0 \
  b2eb93b8ddab8c228993567947bca1e133736980c22754687874e3896f7d6d0a \
  fb1120b91e48d402fac20faa33880e77bace82c85d6688df0aa99bf5084430e4
EXPORTER_SECRET \
  b2eb93b8ddab8c228993567947bca1e133736980c22754687874e3896f7d6d0a \
  db1f4fa1a6942fb125d4cc47e02938b6f8030c6956bb81b9e3269f1cf855a8f8
```

Note that secrets from the two connections might be interleaved as shown here, because secrets could be logged as they are generated.

The following shows a log entry for a TLS 1.2 connection.

```
# NOTE: '\' line wrapping per RFC 8792

CLIENT_RANDOM \
  ad52329fcadd34ee3aa07092680287f09954823e26d7b5ae25c0d47714152a6a \
  97af4c8618cfdc0b2326e590114c2ec04b43b08b7e2c3f8124cc61a3b068ba966\
  9517e744e3117c3ce6c538a2d88dfdf
```

Acknowledgments

The SSLKEYLOGFILE format originated in the NSS project, but it has evolved over time as TLS has changed. Many people contributed to this evolution. The author is only documenting the format as it is used.

Author's Address

Martin Thomson
Mozilla
Email: mt@lowentropy.net

wish
Internet-Draft
Updates: 8842, 8840 (if approved)
Intended status: Standards Track
Expires: 4 November 2024

S. Murillo
Millicast
A. Gouaillard
CoSMo Software

3 May 2024

WebRTC-HTTP ingestion protocol (WHIP)
draft-ietf-wish-whip-14

Abstract

This document describes a simple HTTP-based protocol that will allow WebRTC-based ingestion of content into streaming services and/or CDNs.

This document updates RFC 8842 and RFC 8840.

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 4 November 2024.

Copyright Notice

Table of Contents

1. Introduction

The IETF RTCWEB working group standardized JSEP ([RFC9429]), a mechanism used to control the setup, management, and teardown of a multimedia session. It also describes how to negotiate media flows using the Offer/Answer Model with the Session Description Protocol (SDP) [RFC3264] including the formats for data sent over the wire (e.g., media types, codec parameters, and encryption). WebRTC intentionally does not specify a signaling transport protocol at

application level.

Unfortunately, the lack of a standardized signaling mechanism in WebRTC has been an obstacle to adoption as an ingestion protocol within the broadcast/streaming industry, where a streamlined production pipeline is taken for granted: plug in cables carrying raw media to hardware encoders, then push the encoded media to any streaming service or Content Delivery Network (CDN) ingest using an ingestion protocol.

While WebRTC can be integrated with standard signaling protocols like SIP [RFC3261] or XMPP [RFC6120], they are not designed to be used in broadcasting/streaming services, and there is also no sign of adoption in that industry. RTSP [RFC7826], which is based on RTP, does not support the SDP offer/answer model [RFC3264] for negotiating the characteristics of the media session.

This document proposes a simple protocol based on HTTP for supporting WebRTC as media ingestion method which:

* Is easy to implement,

* Is as easy to use as popular IP-based broadcast protocols

* Is fully compliant with WebRTC and RTCWEB specs

* Enables ingestion on both traditional media platforms and WebRTC end-to-end platforms, achieving the lowest possible latency.

* Lowers the requirements on both hardware encoders and broadcasting services to support WebRTC.

* Is usable both in web browsers and in standalone encoders.

2. Terminology

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. Overview

The WebRTC-HTTP Ingest Protocol (WHIP) is designed to facilitate a one-time exchange of Session Description Protocol (SDP) offers and answers using HTTP POST requests. This exchange is a fundamental step in establishing an Interactive Connectivity Establishment (ICE) and Datagram Transport Layer Security (DTLS) session between the WHIP client, which represents the encoder or media producer, and the media server, the broadcasting ingestion endpoint.

Upon successful establishment of the ICE/DTLS session, unidirectional media data transmission commences from the WHIP client to the media server. It is important to note that SDP renegotiations are not supported in WHIP, meaning that no modifications to the "m=" sections can be made after the initial SDP offer/answer exchange via HTTP POST is completed and only ICE related information can be updated via HTTP PATCH requests as defined in Section 4.1.

The following diagram illustrates the core operation of the WHIP protocol for initiating and terminating an ingest session:

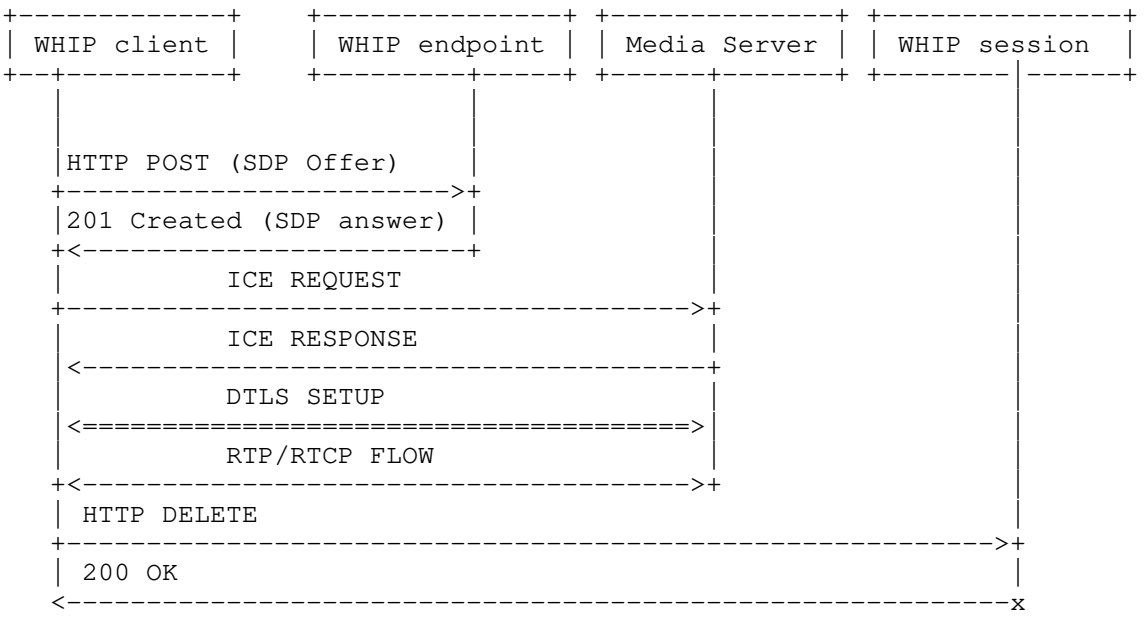


Figure 1: WHIP session setup and teardown

The elements in Figure 1 are described as follows:

* WHIP client: This represents the WebRTC media encoder or producer, which functions as a client of the WHIP protocol by encoding and delivering media to a remote media server.

* WHIP endpoint: This denotes the ingest server that receives the initial WHIP request.

* WHIP endpoint URL: Refers to the URL of the WHIP endpoint responsible for creating the WHIP session.

* Media server: This is the WebRTC media server or consumer responsible for establishing the media session with the WHIP client and receiving the media content it produces.

* WHIP session: Indicates the allocated HTTP resource by the WHIP endpoint for handling an ongoing ingest session.

* WHIP session URL: Refers to the URL of the WHIP resource allocated by the WHIP endpoint for a specific media session. The WHIP client can send requests to the WHIP session using this URL to modify the session, such as ICE operations or termination.

The Figure 1 illustrates the communication flow between a WHIP client, WHIP endpoint, media server, and WHIP session. This flow outlines the process of setting up and tearing down a ingestion session using the WHIP protocol, involving negotiation, ICE for Network Address Translation (NAT) traversal, DTLS for security, and RTP/RTCP for media transport:

* WHIP client: Initiates the communication by sending an HTTP POST with an SDP Offer to the WHIP endpoint.

* WHIP endpoint: Responds with a "201 Created" message containing an SDP answer.

* WHIP client and media server: Establish an ICE and DTLS sessions for NAT traversal and secure communication.

* RTP/RTCP Flow: Real-time Transport Protocol and Real-time Transport Control Protocol flows are established for media transmission from the WHIP client to the media server

* WHIP client: Sends an HTTP DELETE to terminate the WHIP session.

* WHIP session: Responds with a "200 OK" to confirm the session termination.

4. Protocol Operation

In order to set up an ingestion session, the WHIP client MUST generate an SDP offer according to the JSEP rules for an initial offer as in Section 5.2.1 of [RFC9429] and perform an HTTP POST request as per Section 9.3.3 of [RFC9110] to the configured WHIP endpoint URL.

The HTTP POST request MUST have a content type of "application/sdp" and contain the SDP offer as the body. The WHIP endpoint MUST generate an SDP answer according to the JSEP rules for an initial answer as in Section 5.3.1 of [RFC9429] and return a "201 Created" response with a content type of "application/sdp", the SDP answer as the body, and a Location header field pointing to the newly created WHIP session.

As the WHIP protocol only supports the ingestion use case with unidirectional media, the WHIP client SHOULD use "sendonly" attribute in the SDP offer but MAY use the "sendrecv" attribute instead, "inactive" and "recvonly" attributes MUST NOT be used. The WHIP endpoint MUST use "recvonly" attribute in the SDP answer.

If the HTTP POST to the WHIP endpoint has a content type different than "application/sdp", the WHIP endpoint MUST reject the HTTP POST request with a "415 Unsupported Media Type" error response. If the SDP body is malformed, the WHIP session MUST reject the HTTP POST with a "400 Bad Request" error response.

Following Figure 2 is an example of an HTTP POST sent from a WHIP client to a WHIP endpoint and the "201 Created" response from the WHIP endpoint containing the Location header pointing to the newly created WHIP session:

```
POST /whip/endpoint HTTP/1.1
Host: whip.example.com
Content-Type: application/sdp
Content-Length: 1101

v=0
o=- 5228595038118931041 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE 0 1
a=extmap-allow-mixed
a=ice-options:trickle ice2
m=audio 9 UDP/TLS/RTP/SAVPF 111
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
```

```
a=ice-ufrag:EsAw
a=ice-pwd:bP+XJMM09aR8AiX1jdukzR6Y
a=fingerprint:sha-256 DA:7B:57:DC:28:CE:04:4F:31:79:85:C4:31:67:EB:27:58:29:ED:77
:2A:0D:24:AE:ED:AD:30:BC:BD:F1:9C:02
a=setup:actpass
a=mid:0
a=extmap:4 urn:ietf:params:rtp-hdrext:sdes:mid
a=sendonly
a=msid:d46fb922-d52a-4e9c-aa87-444eadc1521b ce326ecf-a081-453a-8f9f-0605d5ef4128
a=rtcp-mux
a=rtcp-mux-only
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10;useinbandfec=1
m=video 0 UDP/TLS/RTP/SAVPF 96 97
a=mid:1
a=bundle-only
a=extmap:4 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:10 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=extmap:11 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-id
a=sendonly
a=msid:d46fb922-d52a-4e9c-aa87-444eadc1521b 3956b460-40f4-4d05-acef-03abcdd8c6fd
a=rtpmap:96 VP8/90000
a=rtcp-fb:96 ccm fir
a=rtcp-fb:96 nack
a=rtcp-fb:96 nack pli
a=rtpmap:97 rtx/90000
a=fmtp:97 apt=96

HTTP/1.1 201 Created
ETag: "xyzzy"
Content-Type: application/sdp
Content-Length: 1053
Location: https://whip.example.com/session/id

v=0
o=- 1657793490019 1 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE 0 1
a=extmap-allow-mixed
a=ice-lite
a=ice-options:trickle ice2
m=audio 9 UDP/TLS/RTP/SAVPF 111
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=ice-ufrag:38sdf4fdsf54
a=ice-pwd:2e13dde17c1cb009202f627fab90cbec358d766d049c9697
a=fingerprint:sha-256 F7:EB:F3:3E:AC:D2:EA:A7:C1:EC:79:D9:B3:8A:35:DA:70:86:4F:46
:D9:2D:CC:D0:BC:81:9F:67:EF:34:2E:BD
a=candidate:1 1 UDP 2130706431 198.51.100.1 39132 typ host
```

```
a=setup:passive
a=mid:0
a=extmap:4 urn:ietf:params:rtp-hdrext:sdes:mid
a=recvonly
a=rtcp-mux
a=rtcp-mux-only
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10;useinbandfec=1
m=video 0 UDP/TLS/RTP/SAVPF 96 97
c=IN IP4 0.0.0.0
a=mid:1
a=bundle-only
a=extmap:4 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:10 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=extmap:11 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-id
a=recvonly
a=rtpmap:96 VP8/90000
a=rtcp-fb:96 ccm fir
a=rtcp-fb:96 nack
a=rtcp-fb:96 nack pli
a=rtpmap:97 rtx/90000
a=fmtp:97 apt=96
```

Figure 2: Example of SDP offer/answer exchange done via an HTTP POST

Once a session is setup, consent freshness as per [RFC7675] SHALL be used to detect non-graceful disconnection by full ICE implementations and DTLS teardown for session termination by either side.

To explicitly terminate a WHIP session, the WHIP client MUST perform an HTTP DELETE request to the WHIP session URL returned in the Location header field of the initial HTTP POST. Upon receiving the HTTP DELETE request, the WHIP session will be removed and the resources freed on the media server, terminating the ICE and DTLS sessions.

A media server terminating a session MUST follow the procedures in Section 5.2 of [RFC7675] for immediate revocation of consent.

The WHIP endpoints MUST return an "405 Method Not Allowed" response for any HTTP request method different than OPTIONS and POST on the endpoint URL in order to reserve their usage for future versions of this protocol specification.

The WHIP endpoints MUST support OPTIONS requests for Cross-Origin Resource Sharing (CORS) as defined in [FETCH]. The "200 OK" response to any OPTIONS request SHOULD include an "Accept-Post" header with a media type value of "application/sdp" as per [W3C.REC-ldp-20150226].

The WHIP sessions MUST return an "405 Method Not Allowed" response for any HTTP request method different than PATCH and DELETE on the session URLs in order to reserve their usage for future versions of this protocol specification.

4.1. ICE support

ICE [RFC8845] is a protocol addressing the complexities of NAT traversal, commonly encountered in Internet communication. NATs hinder direct communication between devices on different local networks, posing challenges for real-time applications. ICE facilitates seamless connectivity by employing techniques to discover and negotiate efficient communication paths.

Trickle ICE [RFC8838] optimizes the connectivity process by incrementally sharing potential communication paths, reducing latency, and facilitating quicker establishment.

ICE Restarts are crucial for maintaining connectivity in dynamic network conditions or disruptions, allowing devices to re-establish communication paths without complete renegotiation. This ensures minimal latency and reliable real-time communication.

Trickle ICE and ICE restart support are RECOMMENDED for both WHIP sessions and clients.

4.1.1. HTTP PATCH request usage

The WHIP client MAY perform trickle ICE or ICE restarts by sending an HTTP PATCH request as per [RFC5789] to the WHIP session URL, with a body containing a SDP fragment with media type "application/trickle-ice-sdpfrag" as specified in [RFC8840] carrying the relevant ICE information.

If the HTTP PATCH to the WHIP session has a content type different than "application/trickle-ice-sdpfrag", the WHIP session MUST reject the HTTP PATCH request with a "415 Unsupported Media Type" error response. If the SDP fragment is malformed, the WHIP session MUST reject the HTTP PATCH with a "400 Bad Request" error response.

If the WHIP session supports either Trickle ICE or ICE restarts, but not both, it MUST return a "422 Unprocessable Content" response for the HTTP PATCH requests that are not supported as per Section 15.5.21 of [RFC9110].

The WHIP client MAY send overlapping HTTP PATCH requests to one WHIP session. Consequently, as those HTTP PATCH requests may be received out-of-order by the WHIP session, if WHIP session supports ICE

restarts, it MUST generate a unique strong entity-tag identifying the ICE session as per Section 8.8.3 of [RFC9110], being OPTIONAL otherwise. The initial value of the entity-tag identifying the initial ICE session MUST be returned in an ETag header field in the "201 Created" response to the initial POST request to the WHIP endpoint.

WHIP clients SHOULD NOT use entity-tag validation when matching a specific ICE session is not required, such as for example when initiating a DELETE request to terminate a session. WHIP sessions MUST ignore any entity-tag value sent by the WHIP client when ICE session matching is not required, as in the HTTP DELETE request.

4.1.2. Trickle ICE

Depending on the Trickle ICE support on the WHIP client, the initial offer by the WHIP client MAY be sent after the full ICE gathering is complete with the full list of ICE candidates, or it MAY only contain local candidates (or even an empty list of candidates) as per [RFC8845]. For the purpose of reducing setup times, when using Trickle ICE the WHIP client SHOULD send the SDP offer as soon as possible, containing either locally gathered ICE candidates or an empty list of candidates.

In order to simplify the protocol, the WHIP session cannot signal additional ICE candidates to the WHIP client after the SDP answer has been sent. The WHIP endpoint SHALL gather all the ICE candidates for the media server before responding to the client request and the SDP answer SHALL contain the full list of ICE candidates of the media server.

As the WHIP client needs to know the WHIP session URL associated with the ICE session in order to send a PATCH request containing new ICE candidates, it MUST wait and buffer any gathered candidates until the "201 Created" HTTP response to the initial POST request is received. In order to lower the HTTP traffic and processing time required the WHIP client SHOULD send a single aggregated HTTP PATCH request with all the buffered ICE candidates once the response is received. Additionally, if ICE restarts are supported by the WHIP session, the WHIP client needs to know the entity-tag associated with the ICE session in order to send a PATCH request containing new ICE candidates, so it MUST also wait and buffer any gathered candidates until it receives the HTTP response with the new entity-tag value to the last PATCH request performing an ICE restart.

WHIP clients generating the HTTP PATCH body with the SDP fragment and its subsequent processing by WHIP sessions MUST follow to the guidelines defined in Section 4.4 of [RFC8840] with the following considerations:

* As per [RFC9429], only m-sections not marked as bundle-only can gather ICE candidates, so given that the "max-bundle" policy is being used, the SDP fragment will contain only the offerer-tagged m-line of the bundle group.

* The WHIP client MAY exclude ICE candidates from the HTTP PATCH body if they have already been confirmed by the WHIP session with a successful HTTP response to a previous HTTP PATCH request.

If the WHIP session is using entity-tags for identifying the ICE sessions in explained in Section 4.1.1, a WHIP client sending a PATCH request for performing trickle ICE MUST include an "If-Match" header field with the latest known entity-tag as per Section 13.1.1 of [RFC9110]. When the PATCH request is received by the WHIP session, it MUST compare the indicated entity-tag value with the current entity-tag of the resource as per Section 13.1.1 of [RFC9110] and return a "412 Precondition Failed" response if they do not match. If the HTTP PATCH request does not contain an "If-Match" header the WHIP session MUST return an "428 Precondition Required" response as per Section 3 of [RFC6585].

When a WHIP session receives a PATCH request that adds new ICE candidates without performing an ICE restart, it MUST return a "204 No Content" response without a body and MUST NOT include an ETag header in the response. If the WHIP session does not support a candidate transport or is not able to resolve the connection address, it MUST silently discard the candidate and continue processing the rest of the request normally.

```
PATCH /session/id HTTP/1.1
Host: whip.example.com
If-Match: "xyzzy"
Content-Type: application/trickle-ice-sdpfrag
Content-Length: 576

a=group:BUNDLE 0 1
m=audio 9 UDP/TLS/RTP/SAVPF 111
a=mid:0
a=ice-ufrag:EsAw
a=ice-pwd:P2uYro0UCOQ4zxjKXaWCBui1
a=candidate:1387637174 1 udp 2122260223 192.0.2.1 61764 typ host generation 0 ufr
ag EsAw network-id 1
a=candidate:3471623853 1 udp 2122194687 198.51.100.2 61765 typ host generation 0
ufrag EsAw network-id 2
a=candidate:473322822 1 tcp 1518280447 192.0.2.1 9 typ host tcptype active genera
tion 0 ufrag EsAw network-id 1
a=candidate:2154773085 1 tcp 1518214911 198.51.100.2 9 typ host tcptype active ge
neration 0 ufrag EsAw network-id 2
a=end-of-candidates

HTTP/1.1 204 No Content
```

Figure 3: Example of a Trickle ICE request and response

Figure 3 shows an example of the Trickle ICE procedure where the WHIP client sends a PATCH request with updated ICE candidate information and receives a successful response from the WHIP session.

4.1.3. ICE Restarts

As defined in [RFC8839], when an ICE restart occurs, a new SDP offer/answer exchange is triggered. However, as WHIP does not support renegotiation of non-ICE related SDP information, a WHIP client will not send a new offer when an ICE restart occurs. Instead, the WHIP client and WHIP session will only exchange the relevant ICE information via an HTTP PATCH request as defined in Section 4.1.1 and MUST assume that the previously negotiated non-ICE related SDP information still apply after the ICE restart.

When performing an ICE restart, the WHIP client MUST include the updated "ice-pwd" and "ice-ufrag" in the SDP fragment of the HTTP PATCH request body as well as the new set of gathered ICE candidates as defined in [RFC8840]. Similar what is defined in Section 4.1.2, as per [RFC9429] only m-sections not marked as bundle-only can gather ICE candidates, so given that the "max-bundle" policy is being used, the SDP fragment will contain only the offerer-tagged m-line of the bundle group. A WHIP client sending a PATCH request for performing ICE restart MUST contain an "If-Match" header field with a field-value "*" as per Section 13.1.1 of [RFC9110].

[RFC8840] states that an agent MUST discard any received requests containing "ice-pwd" and "ice-ufrag" attributes that do not match those of the current ICE Negotiation Session, however, any WHIP session receiving an updated "ice-pwd" and "ice-ufrag" attributes MUST consider the request as performing an ICE restart instead and, if supported, SHALL return a "200 OK" with an "application/trickle-ice-sdpfrag" body containing the new ICE username fragment and password and a new set of ICE candidates for the WHIP session. Also, the "200 OK" response for a successful ICE restart MUST contain the new entity-tag corresponding to the new ICE session in an ETag response header field and MAY contain a new set of ICE candidates for the media server.

As defined in Section 4.4.1.1.1 of [RFC8839] the set of candidates after an ICE restart may include some, none, or all of the previous candidates for that data stream and may include a totally new set of candidates. So after performing a successful ICE restart, both the WHIP client and the WHIP session MUST replace the previous set of remote candidates with the new set exchanged in the HTTP PATCH request and response, discarding any remote ICE candidate not present on the new set. Both the WHIP client and the WHIP session MUST ensure that the HTTP PATCH requests and response bodies include the same 'ice-options,' 'ice-pacing,' and 'ice-lite' attributes as those used in the SDP offer or answer.

If the ICE restart request cannot be satisfied by the WHIP session, the resource MUST return an appropriate HTTP error code and MUST NOT terminate the session immediately and keep the existing ICE session. The WHIP client MAY retry performing a new ICE restart or terminate the session by issuing an HTTP DELETE request instead. In any case, the session MUST be terminated if the ICE consent expires as a consequence of the failed ICE restart as per Section 5.1 of [RFC7675].

In case of unstable network conditions, the ICE restart HTTP PATCH requests and responses might be received out of order. In order to mitigate this scenario, when the client performs an ICE restart, it MUST discard any previous ICE username and passwords fragments and ignore any further HTTP PATCH response received from a pending HTTP PATCH request. WHIP clients MUST apply only the ICE information received in the response to the last sent request. If there is a mismatch between the ICE information at the WHIP client and at the WHIP session (because of an out-of-order request), the STUN requests will contain invalid ICE information and will be dropped by the receiving side. If this situation is detected by the WHIP client, it MUST send a new ICE restart request to the server.

```
PATCH /session/id HTTP/1.1
Host: whip.example.com
If-Match: "*"
Content-Type: application/trickle-ice-sdpfrag
Content-Length: 82

a=ice-options:trickle ice2
a=group:BUNDLE 0 1
m=audio 9 UDP/TLS/RTP/SAVPF 111
a=mid:0
a=ice-ufrag:ysXw
a=ice-pwd:vw5LmwG4y/e6dPP/zAP9Gp5k
a=candidate:1387637174 1 udp 2122260223 192.0.2.1 61764 typ host generation 0 ufr
ag EsAw network-id 1
a=candidate:3471623853 1 udp 2122194687 198.51.100.2 61765 typ host generation 0
ufrag EsAw network-id 2
a=candidate:473322822 1 tcp 1518280447 192.0.2.1 9 typ host tcptype active genera
tion 0 ufrag EsAw network-id 1
a=candidate:2154773085 1 tcp 1518214911 198.51.100.2 9 typ host tcptype active ge
neration 0 ufrag EsAw network-id 2

HTTP/1.1 200 OK
ETag: "abccd"
Content-Type: application/trickle-ice-sdpfrag
Content-Length: 252

a=ice-lite
a=ice-options:trickle ice2
a=group:BUNDLE 0 1
m=audio 9 UDP/TLS/RTP/SAVPF 111
a=mid:0
a=ice-ufrag:289b31b754eaa438
a=ice-pwd:0b66f472495ef0ccac7bda653ab6be49ea13114472a5d10a
a=candidate:1 1 udp 2130706431 198.51.100.1 39132 typ host
a=end-of-candidates
```

Figure 4: Example of an ICE restart request and response

Figure 3 demonstrates a Trickle ICE restart procedure example. The WHIP client sends a PATCH request containing updated ICE information, including a new ufrag and password, along with newly gathered ICE candidates. In response, the WHIP session provides ICE information for the session after the ICE restart, including the updated ufrag and password, as well as the previous ICE candidate.

4.2. WebRTC constraints

To simplify the implementation of WHIP in both clients and media servers, WHIP introduces specific restrictions on WebRTC usage. The following subsections will explain these restrictions in detail:

4.2.1. SDP Bundle

Both the WHIP client and the WHIP endpoint SHALL support [RFC9143] and use "max-bundle" policy as defined in [RFC9429]. The WHIP client and the media server MUST support multiplexed media associated with the BUNDLE group as per Section 9 of [RFC9143]. In addition, per [RFC9143] the WHIP client and media server SHALL use RTP/RTCP multiplexing for all bundled media. In order to reduce the network resources required at the media server, both The WHIP client and WHIP endpoints MUST include the "rtcp-mux-only" attribute in each bundled "m=" sections as per Section 3 of [RFC8858].

4.2.2. Single MediaStream

WHIP only supports a single MediaStream as defined in [RFC8830] and therefore all "m=" sections MUST contain an "msid" attribute with the same value. The MediaStream MUST contain at least one MediaStreamTrack of any media kind and it MUST NOT have two or more than MediaStreamTracks for the same media (audio or video). However, it would be possible for future revisions of this spec to allow more than a single MediaStream or MediaStreamTrack of each media kind, so in order to ensure forward compatibility, if the number of audio and or video MediaStreamTracks or number of MediaStreams are not supported by the WHIP endpoint, it MUST reject the HTTP POST request with a "406 Not Acceptable" error response.

4.2.3. No partially successful answers

The WHIP endpoint SHOULD NOT reject individual "m=" sections as per Section 5.3.1 of [RFC9429] in case there is any error processing the "m=" section, but reject the HTTP POST request with a "406 Not Acceptable" error response to prevent having partially successful ingest sessions which can be misleading to end users.

4.2.4. DTLS setup role and SDP "setup" attribute

When a WHIP client sends an SDP offer, it SHOULD insert an SDP "setup" attribute with an "actpass" attribute value, as defined in [RFC8842]. However, if the WHIP client only implements the DTLS client role, it MAY use an SDP "setup" attribute with an "active" attribute value. If the WHIP endpoint does not support an SDP offer with an SDP "setup" attribute with an "active" attribute value, it SHOULD reject the request with a "422 Unprocessable Entity" response.

NOTE: [RFC8842] defines that the offerer must insert an SDP "setup" attribute with an "actpass" attribute value. However, the WHIP client will always communicate with a media server that is expected to support the DTLS server role, in which case the client might choose to only implement support for the DTLS client role.

4.2.5. Trickle ICE and ICE restarts

The media server SHOULD support full ICE, unless it is connected to the Internet with an IP address that is accessible by each WHIP client that is authorized to use it, in which case it MAY support only ICE lite. The WHIP client MUST implement and use full ICE.

Trickle ICE and ICE restarts support is OPTIONAL for both the WHIP clients and media servers as explained in Section 4.1.

4.3. Load balancing and redirections

WHIP endpoints and media servers might not be colocated on the same server, so it is possible to load balance incoming requests to different media servers.

WHIP clients SHALL support HTTP redirections as per Section 15.4 of [RFC9110]. In order to avoid POST requests to be redirected as GET requests, status codes 301 and 302 MUST NOT be used and the preferred method for performing load balancing is via the "307 Temporary Redirect" response status code as described in Section 15.4.8 of [RFC9110]. Redirections are not required to be supported for the PATCH and DELETE requests.

In case of high load, the WHIP endpoints MAY return a "503 Service Unavailable" response indicating that the server is currently unable to handle the request due to a temporary overload or scheduled maintenance as described in Section 15.6.4 of [RFC9110], which will likely be alleviated after some delay. The WHIP endpoint might send a Retry-After header field indicating the minimum time that the user agent ought to wait before making a follow-up request as described in Section 10.2.3 of [RFC9110].

4.4. STUN/TURN server configuration

The WHIP endpoint MAY return STUN/TURN server configuration URLs and credentials usable by the client in the "201 Created" response to the HTTP POST request to the WHIP endpoint URL.

A reference to each STUN/TURN server will be returned using the "Link" header field [RFC8288] with a "rel" attribute value of "ice-server". The Link target URI is the server URI as defined in [RFC7064] and [RFC7065]. The credentials are encoded in the Link target attributes as follows:

* username: If the Link header field represents a TURN server, and credential-type is "password", then this attribute specifies the username to use with that TURN server.

* credential: If the "credential-type" attribute is missing or has a "password" value, the credential attribute represents a long-term authentication password, as described in Section 9.2 of [RFC8489].

* credential-type: If the Link header field represents a TURN server, then this attribute specifies how the credential attribute value should be used when that TURN server requests authorization. The default value if the attribute is not present is "password".

```
Link: <stun:stun.example.net>; rel="ice-server"
Link: <turn:turn.example.net?transport=udp>; rel="ice-server";
      username="user"; credential="myPassword"; credential-type="password"
Link: <turn:turn.example.net?transport=tcp>; rel="ice-server";
      username="user"; credential="myPassword"; credential-type="password"
Link: <turns:turn.example.net?transport=tcp>; rel="ice-server";
      username="user"; credential="myPassword"; credential-type="password"
```

Figure 5: Example of a STUN/TURN servers configuration

Figure 5 illustrates the Link headers included in a 201 Created response, providing the ICE server URLs and associated credentials.

NOTE: The naming of both the "rel" attribute value of "ice-server" and the target attributes follows the one used on the W3C WebRTC recommendation [W3C.REC-webrtc-20210126] RTCConfiguration dictionary in section 4.2.1. "rel" attribute value of "ice-server" is not prepended with the "urn:ietf:params:whip:" so it can be reused by other specifications which may use this mechanism to configure the usage of STUN/TURN servers.

NOTE: Depending on the ICE Agent implementation, the WHIP client may need to call the setConfiguration method before calling the setLocalDescription method with the local SDP offer in order to avoid having to perform an ICE restart for applying the updated STUN/TURN server configuration on the next ICE gathering phase.

There are some WebRTC implementations that do not support updating the STUN/TURN server configuration after the local offer has been created as specified in Section 4.1.18 of [RFC9429]. In order to support these clients, the WHIP endpoint MAY also include the STUN/TURN server configuration on the responses to OPTIONS request sent to the WHIP endpoint URL before the POST request is sent. However, this method is not NOT RECOMMENDED to be used by the WHIP clients and, if supported by the underlying WHIP client's webrtc implementation, the WHIP client SHOULD wait for the information to be returned by the WHIP endpoint on the response of the HTTP POST request instead.

The generation of the TURN server credentials may require performing a request to an external provider, which can both add latency to the OPTIONS request processing and increase the processing required to handle that request. In order to prevent this, the WHIP endpoint SHOULD NOT return the STUN/TURN server configuration if the OPTIONS request is a preflight request for CORS as defined in [FETCH], that is, if The OPTIONS request does not contain an Access-Control-Request-Method with "POST" value and the the Access-Control-Request-Headers HTTP header does not contain the "Link" value.

The WHIP clients MAY also support configuring the STUN/TURN server URIs with long term credentials provided by either the broadcasting service or an external TURN provider, overriding the values provided by the WHIP endpoint.

4.5. Authentication and authorization

All WHIP endpoints, sessions and clients MUST support HTTP Authentication as per Section 11 of [RFC9110] and in order to ensure interoperability, bearer token authentication as defined in the next section MUST be supported by all WHIP entities. However this does not preclude the support of additional HTTP authentication schemes as defined in Section 11.6 of [RFC9110].

4.5.1. Bearer token authentication

WHIP endpoints and sessions MAY require the HTTP request to be authenticated using an HTTP Authorization header field with a Bearer token as specified in Section 2.1 of [RFC6750]. WHIP clients MUST implement this authentication and authorization mechanism and send the HTTP Authorization header field in all HTTP requests sent to either the WHIP endpoint or session except the preflight OPTIONS requests for CORS.

The nature, syntax, and semantics of the bearer token, as well as how to distribute it to the client, is outside the scope of this document. Some examples of the kind of tokens that could be used

are, but are not limited to, JWT tokens as per [RFC6750] and [RFC8725] or a shared secret stored on a database. The tokens are typically made available to the end user alongside the WHIP endpoint URL and configured on the WHIP clients (similar to the way RTMP URLs and Stream Keys are distributed).

WHIP endpoints and sessions could perform the authentication and authorization by encoding an authentication token within the URLs for the WHIP endpoints or sessions instead. In case the WHIP client is not configured to use a bearer token, the HTTP Authorization header field must not be sent in any request.

4.6. Simulcast and scalable video coding

Simulcast as per [RFC8853] MAY be supported by both the media servers and WHIP clients through negotiation in the SDP offer/answer.

If the client supports simulcast and wants to enable it for ingesting, it MUST negotiate the support in the SDP offer according to the procedures in Section 5.3 of [RFC8853]. A server accepting a simulcast offer MUST create an answer according to the procedures in Section 5.3.2 of [RFC8853].

It is possible for both media servers and WHIP clients to support Scalable Video Coding (SVC). However, as there is no universal negotiation mechanism in SDP for SVC, the encoder must consider the negotiated codec(s), intended usage, and SVC support in available decoders when configuring SVC.

4.7. Protocol extensions

In order to support future extensions to be defined for the WHIP protocol, a common procedure for registering and announcing the new extensions is defined.

Protocol extensions supported by the WHIP sessions MUST be advertised to the WHIP client in the "201 Created" response to the initial HTTP POST request sent to the WHIP endpoint. The WHIP endpoint MUST return one "Link" header field for each extension that it supports, with the extension "rel" attribute value containing the extension URN and the URL for the HTTP resource that will be available for receiving requests related to that extension.

Protocol extensions are optional for both WHIP clients and servers. WHIP clients MUST ignore any Link attribute with an unknown "rel" attribute value and WHIP session MUST NOT require the usage of any of the extensions.

Each protocol extension MUST register a unique "rel" attribute value at IANA starting with the prefix: "urn:ietf:params:whip:ext" as defined in Section 6.3.

For example, considering a potential extension of server-to-client communication using server-sent events as specified in https://html.spec.whatwg.org/multipage/server-sent-events.html#server-sent-events, the URL for connecting to the server-sent event resource for the ingested stream could be returned in the initial HTTP "201 Created" response with a "Link" header field and a "rel" attribute of "urn:ietf:params:whip:ext:example:server-sent-events" (this document does not specify such an extension, and uses it only as an example).

In this theoretical case, the "201 Created" response to the HTTP POST request would look like:

```
HTTP/1.1 201 Created
Content-Type: application/sdp
Location: https://whip.example.com/session/id
Link: <https://whip.ietf.org/publications/213786HF/sse>;
      rel="urn:ietf:params:whip:ext:example:server-sent-events"
```

Figure 6: Example of a WHIP protocol extension

Figure 6 shows an example of a WHIP protocol extension supported by the WHIP session, as indicated in the Link header of the 201 Created response.

5. Security Considerations

This document specifies a new protocol on top of HTTP and WebRTC, thus, security protocols and considerations from related specifications apply to the WHIP specification. These include:

* WebRTC security considerations: [RFC8826]. HTTPS SHALL be used in order to preserve the WebRTC security model.

* Transport Layer Security (TLS): [RFC8446] and [RFC9147].

* HTTP security: Section 11 of [RFC9112] and Section 17 of [RFC9110].

* URI security: Section 7 of [RFC3986].

On top of that, the WHIP protocol exposes a thin new attack surface specific of the REST API methods used within it:

* HTTP POST flooding and resource exhaustion: It would be possible for an attacker in possession of authentication credentials valid for ingesting a WHIP stream to make multiple HTTP POST to the WHIP endpoint. This will force the WHIP endpoint to process the incoming SDP and allocate resources for being able to setup the DTLS/ICE connection. While the malicious client does not need to initiate the DTLS/ICE connection at all, the WHIP session will have to wait for the DTLS/ICE connection timeout in order to release the associated resources. If the connection rate is high enough, this could lead to resource exhaustion on the servers handling the requests and it will not be able to process legitimate incoming ingests. In order to prevent this scenario, WHIP endpoints SHOULD implement a rate limit and avalanche control mechanism for incoming initial HTTP POST requests.

* Insecure direct object references (IDOR) on the WHIP session locations: If the URLs returned by the WHIP endpoint for the WHIP sessions location are easy to guess, it would be possible for an attacker to send multiple HTTP DELETE requests and terminate all the WHIP sessions currently running. In order to prevent this scenario, WHIP endpoints SHOULD generate URLs with enough randomness, using a cryptographically secure pseudorandom number generator following the best practices in Randomness Requirements for Security [RFC4086], and implement a rate limit and avalanche control mechanism for HTTP DELETE requests. The security considerations for Universally Unique IDentifier (UUID) [RFC4122], Section 6 are applicable for generating the WHIP sessions location URL.

* HTTP PATCH flooding: Similar to the HTTP POST flooding, a malicious client could also create a resource exhaustion by sending multiple HTTP PATCH request to the WHIP session, although the WHIP sessions can limit the impact by not allocating new ICE candidates and reusing the existing ICE candidates when doing ICE restarts. In order to prevent this scenario, WHIP endpoints SHOULD implement a rate limit and avalanche control mechanism for incoming HTTP PATCH requests.

6. IANA Considerations

This specification adds a new link relation type and a registry for URN sub-namespaces for WHIP protocol extensions.

6.1. Link Relation Type: ice-server

The link relation type below has been registered by IANA per Section 4.2 of [RFC8288].

Relation Name: ice-server

Description: Conveys the STUN and TURN servers that can be used by an ICE Agent to establish a connection with a peer.

Reference: TBD

6.2. Registration of WHIP URN Sub-namespace and WHIP Registry

IANA is asked to add an entry to the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry and create a sub-namespace for the Registered Parameter Identifier as per [RFC3553]: "urn:ietf:params:whip".

To manage this sub-namespace, IANA is asked to create the "WebRTC-HTTP ingestion protocol (WHIP) URNs" registry, which is used to manage entries within the "urn:ietf:params:whip" namespace. The registry description is as follows:

* Registry name: WebRTC-HTTP ingestion protocol (WHIP) URNs
* Specification: this document (RFC TBD)
* Registration policy: Specification Required
* Repository: See Section Section 6.3
* Index value: See Section Section 6.3

6.3. URN Sub-namespace for WHIP

WHIP endpoint utilizes URNs to identify the supported WHIP protocol extensions on the "rel" attribute of the Link header as defined in Section 4.7.

This section creates and registers an IETF URN Sub-namespace for use in the WHIP specifications and future extensions.

6.3.1. Specification Template

Namespace ID:

* The Namespace ID "whip" has been assigned.

Registration Information:

* Version: 1

* Date: TBD

Declared registrant of the namespace:

* Registering organization: The Internet Engineering Task Force.

* Designated contact: A designated expert will monitor the WHIP public mailing list, "wish@ietf.org".

Declaration of Syntactic Structure:

* The Namespace Specific String (NSS) of all URNs that use the "whip" Namespace ID shall have the following structure: urn:ietf:params:whip:{type}:{name}:{other}.

* The keywords have the following meaning:

  - type: The entity type. This specification only defines the "ext" type.

  - name: A required US-ASCII string that conforms to the URN syntax requirements (see [RFC8141]) and defines a major namespace of a WHIP protocol extension. The value MAY also be an industry name or organization name.

  - other: Any US-ASCII string that conforms to the URN syntax requirements (see [RFC8141]) and defines the sub-namespace (which MAY be further broken down in namespaces delimited by colons) as needed to uniquely identify an WHIP protocol extension.

Relevant Ancillary Documentation:

* None

Identifier Uniqueness Considerations:

* The designated contact shall be responsible for reviewing and enforcing uniqueness.

Identifier Persistence Considerations:

* Once a name has been allocated, it MUST NOT be reallocated for a different purpose.

* The rules provided for assignments of values within a sub-namespace MUST be constructed so that the meanings of values cannot change.

* This registration mechanism is not appropriate for naming values whose meanings may change over time.

Process of Identifier Assignment:

* Namespace with type "ext" (e.g., "urn:ietf:params:whip:ext") is reserved for IETF-approved WHIP specifications.

Process of Identifier Resolution:

* None specified.

Rules for Lexical Equivalence:

* No special considerations; the rules for lexical equivalence specified in [RFC8141] apply.

Conformance with URN Syntax:

* No special considerations.

Validation Mechanism:

* None specified.

Scope:

* Global.

6.4. Registering WHIP Protocol Extensions URNs

This section defines the process for registering new WHIP protocol extensions URNs with IANA in the "WebRTC-HTTP ingestion protocol (WHIP) URNs" registry (see Section 6.3).

A WHIP Protocol Extension URNs is used as a value in the "rel" attribute of the Link header as defined in Section 4.7 for the purpose of signaling the WHIP protocol extensions supported by the WHIP endpoints.

WHIP Protocol Extensions URNs have a "ext" type as defined in Section 6.3.

6.4.1. Registration Procedure

The IETF has created a mailing list, "wish@ietf.org", which can be used for public discussion of WHIP protocol extensions proposals prior to registration. Use of the mailing list is strongly encouraged. The IESG has appointed a designated expert RFC8126 who will monitor the wish@ietf.org mailing list and review registrations.

Registration of new "ext" type URNs (in the namespace "urn:ietf:params:whip:ext") belonging to a WHIP Protocol Extension MUST be documented in a permanent and readily available public specification, in sufficient detail so that interoperability between independent implementations is possible and reviewed by the designated expert as per Section 4.6 of [BCP26] . An RFC is REQUIRED for the registration of new value data types that modify existing properties. An RFC is also REQUIRED for registration of WHIP Protocol Extensions URNs that modify WHIP Protocol Extensions previously documented in an existing RFC.

The registration procedure begins when a completed registration template, defined in the sections below, is sent to iana@iana.org. Decisions made by the designated expert can be appealed to an Applications and Real Time (ART) Area Director, then to the IESG. The normal appeals procedure described in [BCP9] is to be followed.

Once the registration procedure concludes successfully, IANA creates or modifies the corresponding record in the WHIP Protocol Extension registry.

An RFC specifying one or more new WHIP Protocol Extension URNs MUST include the completed registration templates, which MAY be expanded with additional information. These completed templates are intended to go in the body of the document, not in the IANA Considerations section. The RFC MUST include the syntax and semantics of any extension-specific attributes that may be provided in a Link header field advertising the extension.

6.4.2. Guidance for Designated Experts

The Designated Expert (DE) is expected to ascertain the existence of suitable documentation (a specification) as described in RFC8126 and to verify that the document is permanently and publicly available.

The DE is also expected to check the clarity of purpose and use of the requested registration.

Additionally, the DE must verify that any request for one of these registrations has been made available for review and comment within the IETF: the DE will post the request to the WebRTC Ingest Signaling over HTTPS (wish) Working Group mailing list (or a successor mailing list designated by the IESG).

If the request comes from within the IETF, it should be documented in an Internet-Draft. Lastly, the DE must ensure that any other request for a code point does not conflict with work that is active or already published within the IETF.

6.4.3. WHIP Protocol Extension Registration Template

A WHIP Protocol Extension URNs is defined by completing the following template:

* URN: A unique URN for the WHIP Protocol Extension (e.g., "urn:ietf:params:whip:ext:example:server-sent-events").

* Reference: A formal reference to the publicly available specification

* Name: A descriptive name of the WHIP Protocol Extension extension (e.g., "Sender Side events").

* Description: A brief description of the function of the extension, in a short paragraph or two

* Contact information: Contact information for the organization or person making the registration

7. Acknowledgements

The authors wish to thank Lorenzo Miniero, Juliusz Chroboczek, Adam Roach, Nils Ohlmeier, Christer Holmberg, Cameron Elliott, Gustavo Garcia, Jonas Birme, Sandro Gauci, Christer Holmberg and everyone else in the WebRTC community that have provided comments, feedback, text and improvement proposals on the document and contributed early implementations of the spec.

8. References

8.1. Normative References

[BCP26] Best Current Practice 26,
<https://www.rfc-editor.org/info/bcp26>.
At the time of writing, this BCP comprises the following:

Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[BCP9] Best Current Practice 9, <https://www.rfc-editor.org/info/bcp9>. At the time of writing, this BCP comprises the following:

Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, DOI 10.17487/RFC2026, October 1996, <https://www.rfc-editor.org/info/rfc2026>.

Dusseault, L. and R. Sparks, "Guidance on Interoperation and Implementation Reports for Advancement to Draft Standard", BCP 9, RFC 5657, DOI 10.17487/RFC5657, September 2009, <https://www.rfc-editor.org/info/rfc5657>.

Housley, R., Crocker, D., and E. Burger, "Reducing the Standards Track to Two Maturity Levels", BCP 9, RFC 6410, DOI 10.17487/RFC6410, October 2011, <https://www.rfc-editor.org/info/rfc6410>.

Resnick, P., "Retirement of the "Internet Official Protocol Standards" Summary Document", BCP 9, RFC 7100, DOI 10.17487/RFC7100, December 2013, <https://www.rfc-editor.org/info/rfc7100>.

Kolkman, O., Bradner, S., and S. Turner, "Characterization of Proposed Standards", BCP 9, RFC 7127, DOI 10.17487/RFC7127, January 2014, <https://www.rfc-editor.org/info/rfc7127>.

Dawkins, S., "Increasing the Number of Area Directors in an IETF Area", BCP 9, RFC 7475, DOI 10.17487/RFC7475, March 2015, <https://www.rfc-editor.org/info/rfc7475>.

Halpern, J., Ed. and E. Rescorla, Ed., "IETF Stream Documents Require IETF Rough Consensus", BCP 9, RFC 8789, DOI 10.17487/RFC8789, June 2020, <https://www.rfc-editor.org/info/rfc8789>.

Rosen, B., "Responsibility Change for the RFC Series", BCP 9, RFC 9282, DOI 10.17487/RFC9282, June 2022, <https://www.rfc-editor.org/info/rfc9282>.

[FETCH] WHATWG, "Fetch - Living Standard", n.d., <https://fetch.spec.whatwg.org>.

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

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <https://www.rfc-editor.org/rfc/rfc3264>.

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <https://www.rfc-editor.org/rfc/rfc3553>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/rfc/rfc4122>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/rfc/rfc5789>.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <https://www.rfc-editor.org/rfc/rfc6585>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[RFC7064] Nandakumar, S., Salgueiro, G., Jones, P., and M. Petit-Huguenin, "URI Scheme for the Session Traversal Utilities for NAT (STUN) Protocol", RFC 7064, DOI 10.17487/RFC7064, November 2013, <https://www.rfc-editor.org/rfc/rfc7064>.

[RFC7065] Petit-Huguenin, M., Nandakumar, S., Salgueiro, G., and P. Jones, "Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers", RFC 7065, DOI 10.17487/RFC7065, November 2013, <https://www.rfc-editor.org/rfc/rfc7065>.

[RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <https://www.rfc-editor.org/rfc/rfc7675>.

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

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

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

[RFC8489] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", RFC 8489, DOI 10.17487/RFC8489, February 2020, <https://www.rfc-editor.org/rfc/rfc8489>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/rfc/rfc8725>.

[RFC8826] Rescorla, E., "Security Considerations for WebRTC", RFC 8826, DOI 10.17487/RFC8826, January 2021, <https://www.rfc-editor.org/rfc/rfc8826>.

[RFC8830] Alvestrand, H., "WebRTC MediaStream Identification in the Session Description Protocol", RFC 8830, DOI 10.17487/RFC8830, January 2021, <https://www.rfc-editor.org/rfc/rfc8830>.

[RFC8838] Ivov, E., Uberti, J., and P. Saint-Andre, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol", RFC 8838, DOI 10.17487/RFC8838, January 2021, <https://www.rfc-editor.org/rfc/rfc8838>.

[RFC8839] Petit-Huguenin, M., Nandakumar, S., Holmberg, C., Keränen, A., and R. Shpount, "Session Description Protocol (SDP) Offer/Answer Procedures for Interactive Connectivity Establishment (ICE)", RFC 8839, DOI 10.17487/RFC8839, January 2021, <https://www.rfc-editor.org/rfc/rfc8839>.

[RFC8840] Ivov, E., Stach, T., Marocco, E., and C. Holmberg, "A Session Initiation Protocol (SIP) Usage for Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (Trickle ICE)", RFC 8840, DOI 10.17487/RFC8840, January 2021, <https://www.rfc-editor.org/rfc/rfc8840>.

[RFC8842] Holmberg, C. and R. Shpount, "Session Description Protocol (SDP) Offer/Answer Considerations for Datagram Transport Layer Security (DTLS) and Transport Layer Security (TLS)", RFC 8842, DOI 10.17487/RFC8842, January 2021, <https://www.rfc-editor.org/rfc/rfc8842>.

[RFC8845] Duckworth, M., Ed., Pepperell, A., and S. Wenger, "Framework for Telepresence Multi-Streams", RFC 8845, DOI 10.17487/RFC8845, January 2021, <https://www.rfc-editor.org/rfc/rfc8845>.

[RFC8853] Burman, B., Westerlund, M., Nandakumar, S., and M. Zanaty, "Using Simulcast in Session Description Protocol (SDP) and RTP Sessions", RFC 8853, DOI 10.17487/RFC8853, January 2021, <https://www.rfc-editor.org/rfc/rfc8853>.

[RFC8858] Holmberg, C., "Indicating Exclusive Support of RTP and RTP Control Protocol (RTCP) Multiplexing Using the Session Description Protocol (SDP)", RFC 8858, DOI 10.17487/RFC8858, January 2021, <https://www.rfc-editor.org/rfc/rfc8858>.

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

[RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[RFC9143] Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", RFC 9143, DOI 10.17487/RFC9143, February 2022, <https://www.rfc-editor.org/rfc/rfc9143>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9429] Uberti, J., Jennings, C., and E. Rescorla, Ed., "JavaScript Session Establishment Protocol (JSEP)", RFC 9429, DOI 10.17487/RFC9429, April 2024, <https://www.rfc-editor.org/rfc/rfc9429>.

[W3C.REC-ldp-20150226] Malhotra, A., Ed., Arwe, J., Ed., and S. Speicher, Ed., "Linked Data Platform 1.0", W3C REC REC-ldp-20150226, W3C REC-ldp-20150226, 26 February 2015, <https://www.w3.org/TR/2015/REC-ldp-20150226/>.

8.2. Informative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/rfc/rfc3261>.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <https://www.rfc-editor.org/rfc/rfc6120>.

[RFC7826] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., and M. Stiemerling, Ed., "Real-Time Streaming Protocol Version 2.0", RFC 7826, DOI 10.17487/RFC7826, December 2016, <https://www.rfc-editor.org/rfc/rfc7826>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/rfc/rfc8141>.

[W3C.REC-webrtc-20210126]
Jennings, C., Ed., Boström, H., Ed., and J. Bruaroey, Ed., "WebRTC 1.0: Real-Time Communication Between Browsers", W3C REC REC-webrtc-20210126, W3C REC-webrtc-20210126, 26 January 2021, <https://www.w3.org/TR/2021/REC-webrtc-20210126/>.

Authors' Addresses

Sergio Garcia Murillo
Millicast
Email: sergio.garcia.murillo@cosmosoftware.io

Alexandre Gouaillard
CoSMo Software
Email: alex.gouaillard@cosmosoftware.io

Dynamic Host Configuration
Internet-Draft
Intended status: Standards Track
Expires: 1 October 2023

W. Kumari
Google, LLC
S. Krishnan
R. Asati
Cisco Systems, Inc.
L. Colitti
J. Linkova
Google, LLC

30 March 2023

Registering Self-generated IPv6 Addresses using DHCPv6
draft-wkumari-dhc-addr-notification-07

Abstract

This document defines a method to inform a DHCPv6 server that a device has a self-generated or statically configured address.

About This Document

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

The latest revision of this draft can be found at https://wkumari.github.io/draft-wkumari-dhc-addr-notification/draft-wkumari-dhc-addr-notification.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-wkumari-dhc-addr-notification/.

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

Source for this draft and an issue tracker can be found at https://github.com/wkumari/draft-wkumari-dhc-addr-notification.

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/.

This Internet-Draft will expire on 1 October 2023.

Copyright Notice

Table of Contents

1. Introduction

It is very common operational practice, especially in enterprise networks, to use IPv4 DHCP logs for troubleshooting or security purposes. Examples of this include a helpdesk dealing with a ticket such as "The CEO's laptop cannot connect to the printer"; if the MAC address of the printer is known (for example from an inventory system), the IPv4 address can be retrieved from the DHCP logs and the printer pinged to determine if it is reachable. Another common example is a Security Operations team discovering suspicious events in outbound firewall logs and then consulting DHCP logs to determine which employee's laptop had that IPv4 address at that time so that they can quarantine it and remove the malware.

This operational practice relies on the DHCP server knowing the IP address assignments. Therefore, the practice does not work if static IP addresses are manually configured on devices or self-assigned addresses (such as when self-configuring an IPv6 address using SLAAC [RFC4862]) are used.

The lack of this parity with IPv4 is one of the reasons that some enterprise networks are unwilling to deploy IPv6.

This document provides a mechanism for a device to inform the DHCPv6 server that it has a self-configured IPv6 address (or has a statically configured address), and thus provides parity with IPv4 in this aspect.

2. Conventions and Definitions

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

3. Description of Mechanism

After successfully assigning a self-generated IPv6 address on one of its interfaces, an end-host implementing this specification SHOULD multicast an ADDR-REG-INFORM message in order to inform the DHCPv6 server that this address is in use.

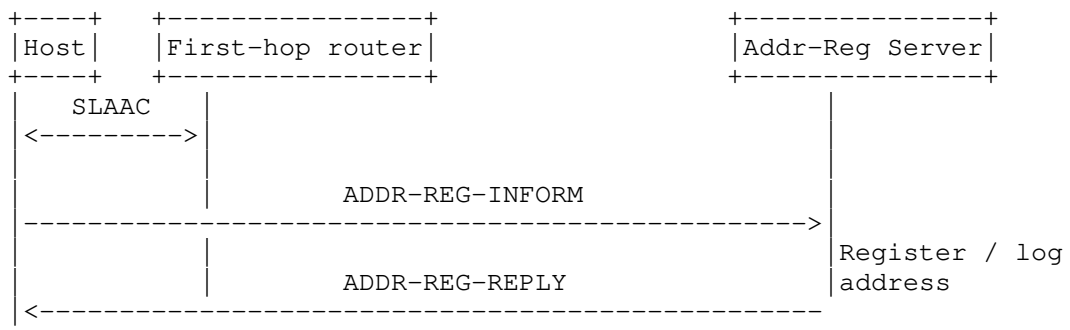
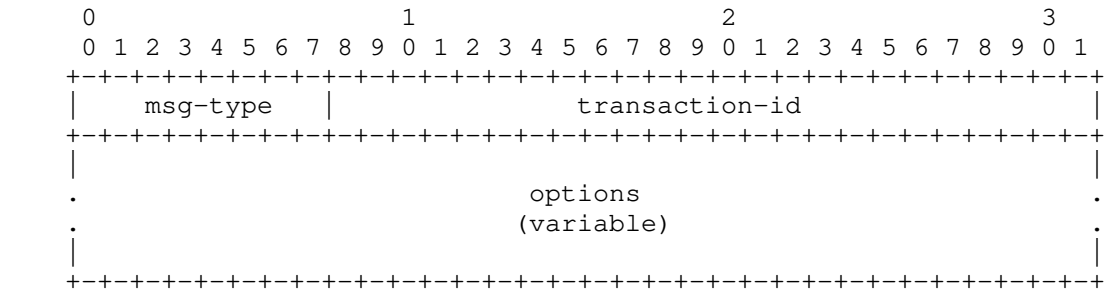


Figure 1: Address Registration Procedure

4. DHCPv6 ADDR-REG-INFORM Message

The DHCPv6 client sends an ADDR-REG-INFORM message to inform that an IPv6 address is in use. The format of the ADDR-REG-INFORM message is described as follows:



msg-type — Identifies the DHCPv6 message type; Set to ADDR-REG-INFORM (TBA1).

transaction-id — The transaction ID for this message exchange.

options — Options carried in this message.

Figure 2: DHCPv6 ADDR-REG-INFORM message

The ADDR-REG-INFORM message MUST NOT contain server-identifier option and MUST contain the IA Address option. The ADDR-REG-INFORM message is dedicated for clients to initiate an address registration request toward an address registration server. Consequently, clients MUST NOT put any Option Request Option(s) in the ADDR-REG-INFORM message. Clients MAY include other options, such as the Client FQDN Option [RFC4704].

Clients MUST discard any received ADDR-REG-INFORM messages.

Servers MUST discard any ADDR-REG-INFORM messages that meet any of the following conditions:

* the address is not appropriate for the link;

* the message does not include a Client Identifier option;

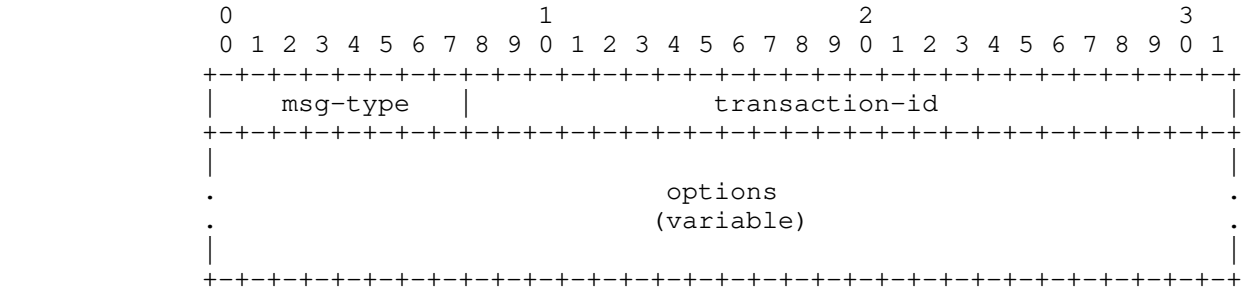
* the message includes a Server Identifier option;

* the message does not include the IA Address option;

* the message includes an Option Request Option.

5. DHCPv6 ADDR-REG-REPLY Message

The DHCPv6 server sends an ADDR-REG-REPLY message in response to a valid ADDR-REG-INFORM message. The format of the ADDR-REG-REPLY message is described as follows:



msg-type — Identifies the DHCPv6 message type; Set to ADDR-REG-REPLY (TBA2).

transaction-id — The transaction ID for this message exchange.

options — Options carried in this message.

Figure 3: DHCPv6 ADDR-REG-REPLY message

The ADDR-REG-INFORM message MUST contain an IA Address option for the address being registered.

Servers MUST ignore any received ADDR-REG-REPLY messages.

The IPv6 destination address of the packet is the address being registered.

6. DHCPv6 Address Registration Procedure

The DHCPv6 protocol is used as the address registration protocol when a DHCPv6 server performs the role of an address registration server. The DHCPv6 IA Address option [RFC8415] is adopted in order to fulfill the address registration interactions.

6.1. DHCPv6 Address Registration Request

The end-host sends a DHCPv6 ADDR-REG-INFORM message to the address registration server to the All_DHCP_Relay_Agents_and_Servers multicast address (ff02::1:2). The host MUST only send the packet on the network interface that has the address being registered (i.e. if the host has multiple interfaces with different addresses, it should only send the packet on the interface with the address being registered). The host MUST send the packet from the address being registered. This is primarily for "fate sharing" purposes - for example, if the network implements some form of L2 security to prevent a client from spoofing other clients' addresses this prevents an attacker from spoofing ADDR-REG-INFORM messages. The host MUST send separate messages for each address being registered.

The end-host MUST include a Client Identifier option in the ADDR-REG-INFORM message.

The host MUST only send the ADDR-REG-INFORM message for valid ([RFC4862]) addresses of global scope ([RFC4007]). The host MUST NOT send the ADDR-REG-INFORM message for addresses configured by DHCPv6.

The host MUST NOT send the ADDR-REG-INFORM message if it has not received any Router Advertisement message with either M or O flags set to 1.

After receiving this ADDR-REG-INFORM message, the address registration server SHOULD verify that the address being registered is "appropriate to the link" as defined by [RFC8415]. If the server believes that address being registered is not appropriate to the link [RFC8415], it MUST drop the message, and SHOULD log this fact. If the address is appropriate, the server:

* SHOULD register or update a binding between the provided Client Identifier and IPv6 address in its database;

* SHOULD log the address registration information (as is done normally for clients which have requested an address), unless configured not to do so;

* SHOULD mark the address as unavailable for use and not include it in future ADVERTISE messages.

* SHOULD send back an ADDR-REG-REPLY message.

If the DHCPv6 server does not support the address registration function, it MUST drop the message, and SHOULD log this fact.

DHCPv6 relay agents and switches that relay address registration messages directly from clients SHOULD include the client's link-layer address in the relayed message using the Client Link-Layer Address option ([RFC6939])

6.2. DHCPv6 Address Registration Acknowledgement

The server SHOULD acknowledge receipt of an ADDR-REG-INFORM message by sending a ADDR-REG-REPLY message back. The ADDR-REG-REPLY message only indicates that the ADDR-REG-INFORM message has been received. It MUST NOT be considered as any indication of the address validity and MUST NOT be required for the address to be usable. DHCPv6 relays, or other devices that snoop ADDR-REG-REPLY messages, MUST NOT add or alter any forwarding or security state based on the ADDR-REG-REPLY message.

6.3. Registration Expiry and Refresh

The client MUST refresh the registration every AddrRegRefresh seconds, where AddrRegRefresh is min(1/3 of the Valid Lifetime filed in the very first PIO received to form the address; 4 hours). Registration refresh packets SHOULD be retransmitted using the same logic as described in the 'Retransmission' section below. In particular, retransmissions SHOULD be jittered to avoid synchronization causing a large number of registrations to expire at the same time.

If the address registration server does not receive such a refresh after the preferred lifetime has passed, it SHOULD remove the record of the Client-Identifier-to-IPv6-address binding.

The client MAY choose to notify the server when an address is no longer being used (the client is disconnecting from the network, the address lifetime expired or the address is being removed from the interface). To indicate that the address is not being used anymore the client MUST set the preferred-lifetime and valid-lifetime fields of the IA Address option to zero.

6.4. Retransmission

To reduce the effects of packet loss on registration, the client SHOULD retransmit the registration message. Retransmissions SHOULD follow the standard retransmission logic specified by section 15 of [RFC8415] with the following default parameters:

* IRT 1 sec

* MRC 3

The client SHOULD allow these parameters to be configured by the administrator.

If an ADDR-REG-REPLY message is received for the address being registered, the client MUST stop retransmission. However, the client can not rely on the server acknowledging receipt of the registration message, because the server might not support address registration.

7. Host configuration

DHCP clients SHOULD allow the administrator to disable sending ADDR-REG-INFORM messages. This could be used, for example, to reduce network traffic on networks where the servers are known not to support the message type. Sending the messages SHOULD be enabled by default.

8. Security Considerations

An attacker may attempt to register a large number of addresses in quick succession in order to overwhelm the address registration server and / or fill up log files. These attacks may be mitigated by using generic DHCPv6 protection such as the AUTH option [RFC8415]. The similar attack vector exist today, e.g. an attacker can DoS the server with messages contained spoofed DUIDs.

If a network is using FCFS SAVI [RFC6620], then the DHCPv6 server can trust that the ADDR-REG-INFORM message was sent by the legitimate owner of the address. This prevents a host from registering an address owned by another host.

One of the use-cases for the mechanism described in this document is to identify sources of malicious traffic after the fact. Note, however, that as the device itself is responsible for informing the DHCPv6 server that it is using an address, a malicious or compromised device can simply not send the ADDR-REG-INFORM message. This is an informational, optional mechanism, and is designed to aid in troubleshooting and forensics. On its own, it is not intended to be a strong security access mechanism.

9. IANA Considerations

This document defines a new DHCPv6 message, the ADDR-REG-INFORM message (TBA1) described in Section 4, that requires an allocation out of the registry of Message Types defined at http://www.iana.org/assignments/dhcpv6-parameters/

10. Normative References

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

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, DOI 10.17487/RFC4007, March 2005, <https://www.rfc-editor.org/rfc/rfc4007>.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, DOI 10.17487/RFC4704, October 2006, <https://www.rfc-editor.org/rfc/rfc4704>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/rfc/rfc4862>.

[RFC6939] Halwasia, G., Bhandari, S., and W. Dec, "Client Link-Layer Address Option in DHCPv6", RFC 6939, DOI 10.17487/RFC6939, May 2013, <https://www.rfc-editor.org/rfc/rfc6939>.

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

[RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/rfc/rfc8415>.

Acknowledgments

Much thanks to Bernie Volz for significant review and feedback, as well as Stuart Cheshire, Alan DeKok, Ryan Globus, Erik Kline, Ted Lemon, Eric Levy-Abegnoli, Mark Smith, Eric Vynke, Timothy Winter for their feedback, comments and guidance.

This document borrows heavily from a previous document, draft-ietf-dhc-addr-registration, which defined "a mechanism to register self-generated and statically configured addresses in DNS through a DHCPv6 server". That document was written Sheng Jiang, Gang Chen, Suresh Krishnan, and Rajiv Asati.

Contributors

Gang Chen
China Mobile
53A, Xibianmennei Ave.
Xuanwu District
Beijing
P.R. China
Email: phdgang@gmail.com

Authors' Addresses

Warren Kumari
Google, LLC
Email: warren@kumari.net

Suresh Krishnan
Cisco Systems, Inc.
Email: suresh.krishnan@gmail.com

Rajiv Asati
Cisco Systems, Inc.
7025 Kit Creek road
Research Triangle Park, 27709-4987
United States of America
Email: rajiva@cisco.com

Lorenzo Colitti
Google, LLC
Shibuya 3-21-3,
Japan
Email: lorenzo@google.com

Jen Linkova
Google, LLC
1 Darling Island Rd
Pyrmont 2009
Australia
Email: furry@google.com