

Network Working Group  
INTERNET-DRAFT  
Category: Informational  
Expires: December 30, 2013

B. Aboba  
M. Thomson  
Skype  
13 June 2013

Emergency Services Support in WebRTC  
draft-aboba-rtcweb-ecrit-01.txt

## Abstract

The Web Real-Time Communication (WebRTC) framework supports interactive communication between web-browsers, including support for audio, video and text. This document describes how emergency services functionality can be implemented within the WebRTC framework, including support for location and call routing as well as interoperability with Public Safety Answering Points (PSAPs) supporting next generation emergency services.

## Legal

THIS DOCUMENT AND THE INFORMATION CONTAINED THEREIN ARE PROVIDED ON AN "AS IS" BASIS AND THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST, AND THE INTERNET ENGINEERING TASK FORCE, DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION THEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## 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 <http://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 December 30, 2013.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	3
1.1 Terminology . . . . .	3
1.2 Prior Work . . . . .	4
2. Location and Call Routing Requirements . . . . .	4
3. Media Requirements . . . . .	7
4. Accessibility . . . . .	8
5. Security Considerations . . . . .	9
6. IANA Considerations . . . . .	11
7. Acknowledgments . . . . .	11
8. References . . . . .	11
8.1. Normative References . . . . .	11
8.2 Informative references . . . . .	11
Authors' Addresses . . . . .	15

## 1. Introduction

The Web Real-Time Communication (WebRTC) framework supports interactive communication between web-browsers, including support for audio, video and text. This document describes how emergency services functionality can be implemented within the WebRTC framework. Since signaling is out of scope of the WebRTC standards suite as noted in "Overview: Real Time Protocols for Browser-based Applications" [I-D.ietf-rtcweb-overview] Section 3, this document focuses on other aspects such as location, call routing and media support.

No guidance is provided as to whether a given WebRTC application or service will be subject to emergency service obligations. As noted in "Best Current Practice for Communications Services in support of Emergency Calling" [RFC6881] Section 4:

Some jurisdictions have regulations governing which devices need to support emergency calling and developers are encouraged to ensure that devices they develop meet relevant regulatory requirements. Unfortunately, the natural variation in those regulations also makes it impossible to accurately describe the cases when developers do or do not have to support emergency calling.

It should also be understood that this document does not advocate use of IP-based communication in all situations. For example, where accurate location cannot be obtained, emergency callers could be better served by utilizing the telephony capabilities of the underlying platform (e.g., a mobile-device) where available, as proposed in [WebTel]. This can enable location to be provided in situations where it would not otherwise be available, as well as permitting an emergency call to be placed even when the device does not have access to the Internet.

The document is laid out as follows: Section 1 provides an introduction and reviews prior work. Section 2 discusses requirements relating to location and call routing. Section 3 discusses media requirements. Section 4 discusses accessibility. Section 5 discusses security considerations.

### 1.1. Terminology

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

This document uses terms from [RFC3261], [RFC5012] and [RFC6443].

## 1.2. Prior Work

The IETF ECRIT WG has developed an overview of the emergency calling architecture as well as a best current practice document detailing implementation requirements.

"Framework for Emergency Calling using Internet Multimedia" [RFC6443] provides an overview of how IETF specifications can be used to support emergency calling using multimedia. At a high level, this involves determination of the caller location, conveyance of the location within a signaling protocol such as Session Initiation Protocol (SIP) [RFC6442], routing of the call using the Location-to-Service Translation (LoST) protocol [RFC5222], and exchange of media using Real-time Transport Protocol (RTP) [RFC3550].

"Best Current Practice for Communication Services in support of Emergency Calling" [RFC6881] builds on [RFC6443] to describe the requirements for end devices ("ED-" requirements), access networks ("AN-"), service providers ("SP-"), Public Safety Answering Points (PSAPs) and intermediate devices ("INT-") to achieve globally interoperable emergency calling on the Internet.

Both [RFC6443] and [RFC6881] assume the use of SIP as the signaling mechanism for emergency calling. As noted in [RFC6443] Section 1:

This document discusses the use of the Session Initiation Protocol (SIP) [RFC3261] by PSAPs and calling parties. While other inter-domain call signaling protocols may be used for emergency calling, SIP is ubiquitous and possesses the proper support of this use case.

Since standardization of signaling is out of scope of the WebRTC standards effort, and WebRTC applications can utilize a wide variety of signaling mechanisms, the requirements described in [RFC6881] do not necessarily apply to WebRTC implementations, applications and services. Therefore in this document, we focus on emergency calling requirements that are independent of the signaling mechanism, such as those relating to accessibility, location, call routing and media.

## 2. Location and Call Routing Requirements

Determination of caller location as well as call routing is an essential aspect of emergency services support. Relevant requirements from [RFC6881] include:

ED-15/INT-4/AN-4 Devices, intermediate Devices and/or access networks SHOULD support a manual method to override the location the access network determines. When the override location is

supplied in civic form, it MUST be possible for the resultant Presence Information Data Format - Location Object (PIDF-LO) received at the PSAP to contain any of the elements specified in [RFC4119] and [RFC5139].

ED-17/INT-9/AN-9 Devices that support endpoint measuring of location MUST have at least a coarse location capability (typically <1km accuracy) for routing of calls. The location mechanism MAY be a service provided by the access network.

ED-24 Where the operating system supporting application programs which need location for emergency calls does not allow access to Layer 2 and Layer 3 functions necessary for a client application to use DHCP location options and/or other location technologies that are specific to the type of access network, the operating system MUST provide a published API conforming to ED-12 through ED-23 and ED-25 through ED-32. It is RECOMMENDED that all operating systems provide such an API.

ED-41/SP-20 Location objects MUST be created with information about the method by which the location was determined, such as GPS, manually entered, or based on access network topology included in a PIDF- LO "method" element. In addition, the source of the location information MUST be included in a PIDF-LO "provided-by" element.

ED-49 Endpoints MUST support one or more mechanisms that allow them to determine their public IP address, for example, STUN [RFC5389].

ED-50 Endpoints MUST support LIS discovery as described in [RFC5986], and the LoST discovery as described in [RFC5223].

Since browser applications do not have direct access to operating system location APIs, ED-24 is not applicable to WebRTC.

For reasons that will be described, automatically obtaining location suitable for emergency use is challenging for WebRTC applications. In order to ensure that location is available when needed, as well as to provide resilience against errors in automated location determination, WebRTC emergency service applications SHOULD support manual override as recommended in ED-15.

The W3C Geolocation API [GeolocationAPI] was not developed with emergency services location in mind, so that requirements ED-17 and ED-41 are not well supported. [GeolocationAPI] does not provide information on the source of the location information as required in ED-41; attempting to infer the source from the accuracy parameter is

NOT RECOMMENDED. Currently, Location Based Services utilized by Geolocation APIs do not warrant their use in emergency services and do not consistently provide the accuracy required by emergency services applications, so that emergency use of the W3C Geolocation API is also NOT RECOMMENDED.

An alternative is to implement location configuration and call routing in Javascript, using an HTTP-based protocol such as HELD [RFC5985] and LoST [RFC5222]. While this approach can provide location usable in emergency services applications, it is only applicable on networks with a Location Information Server (LIS), such as enterprise deployments subject to Multi-Line Telephone System (MLTS) regulations [StateMLTS].

In order to utilize location and call routing services, it is first necessary to locate the appropriate servers. Since the discovery mechanisms described in [RFC5986] and [RFC5223] are based on use of a DHCP option, which cannot be assumed to be accessible in Javascript, ED-50 is difficult to support within WebRTC-based emergency services applications.

For LoST discovery, the emergency services application can determine the appropriate LoST server(s) on its own. To avoid potential issues, it is best to avoid pre-configuration of particular servers, allowing the appropriate server to be determined dynamically.

LIS discovery requires determination of the domain name that can be used for LIS discovery, as noted in [RFC5986] Section 3.4:

If a Device knows one or more alternative domain names that might be used for discovery, it MAY repeat the U-NAPTR process using those domain names as input. For instance, static configuration of a Device might be used to provide a Device with a domain name.

While static configuration of the domain name can be used in situations where device mobility is restricted, the appropriate LIS depends on the network to which the host is attached, so that this is not a general solution.

"Location Information Server (LIS) Discovery using IP address and Reverse DNS" [I.D.ietf-geopriv-res-gw-lis-discovery] specifies a means for a device to discover several alternative domain names that can be used as input to the Dynamic Delegation Discovery Service (DDDS). Since several of the techniques (such as use of PTR RRs and Session Traversal Utilities for NAT (STUN) [RFC5389]) are potentially implementable in WebRTC-based emergency services applications this approach MAY be used.

### 3. Media Requirements

Within [RFC6881] media-related requirements are covered in Section 14. These include:

ED-71 Endpoints MUST send and receive media streams on RTP [RFC3550].

ED-72 Normal SIP offer/answer [RFC3264] negotiations MUST be used to agree on the media streams to be used.

ED-73/SP-41 G.711 A law (and mu Law if they are intended be used in North America) encoded voice as described in [RFC3551] MUST be supported. If the endpoint cannot support G.711, a transcoder MUST be used so that the offer received at the PSAP contains G.711. It is desirable to include wideband codecs such as G.722 and AMR-WB in the offer. PSAPs SHOULD support narrowband codecs common on endpoints in their area to avoid transcoding.

ED-74 Silence suppression (Voice Activity Detection methods) MUST NOT be used on emergency calls. PSAP call takers sometimes get information on what is happening in the background to determine how to process the call.

ED-77 Endpoints supporting video MUST support H.264 per [RFC6184].

Requirement ED-71 is satisfied by compliant WebRTC implementations since "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP" [I-D.ietf-rtcweb-rtp-usage] Section 4.1 requires support for RTP [RFC3550].

Requirement ED-72 is specific to SIP and so does not apply generally to WebRTC implementations, applications and services. However, it is believed that the APIs under development within the W3C WebRTC WG can support this requirement.

Requirement ED-74 is satisfied by compliant WebRTC implementations since the WebRTC APIs under development within W3C [WEBRTC], support silence suppression control via the "constraints" parameter.

[I-D.ietf-rtcweb-rtp-usage] Section 4.3 does not provide a recommendation on a mandatory-to-implement set of codecs. While ED-73 does not require implementation of G.711 if the service supports transcoding, G.711 is not difficult to implement and is widely supported, with a high level of interoperability. Therefore it is recommended that G.711 be included as a mandatory-to-implement audio codec within [I-D.ietf-rtcweb-rtp-usage] Section 4.3.

Currently the disposition of ED-77 is unclear. Discussion of mandatory-to-implement video codecs is ongoing within the IETF RTCWEB WG, but has not reached a conclusion. While there is a need to support interoperable video within emergency services applications, more options may be available within an emergency services context than would be the case for general use. For example, within the PSAP, it may be feasible to support multiple video codecs, either by installation of browser plugins, or by use of multiple browsers. In some emergency service applications (such as the VRS), codec requirements may be specific to the service and may be satisfiable by a custom device or browser approved for use with that service, which may include the required codecs implemented natively or via plug-ins, as the service provider sees fit.

#### 4. Accessibility

By lowering the barriers to development of realtime-enabled browser applications, as well as by building on accessibility support within the browser, WebRTC promises to enable the development of a new generation of accessible emergency applications and services.

In order to support accessibility, it is RECOMMENDED that WebRTC-based emergency applications and services conform to the Web Content Accessibility Guidelines (WCAG) v2.0 [WCAG].

In order to support accessibility for individuals with hearing or speech disabilities, support for textual communications is important.

Currently the W3C is developing a proposed charter for the Timed Text Working Group [TTWG], which will potentially produce a second edition of the timed Text Markup Language (TTML) 1.0 recommendation as well as publishing a recommendation for a version 1.1 specification.

Text-related requirements in [RFC6881] are covered in Section 14, including:

ED-75 Endpoints supporting Instant Messaging (IM) MUST support either [RFC3428] and [RFC4975].

ED-76 Endpoints supporting real-time text MUST use [RFC4103]. The expectations for emergency service support for the real-time text medium are described in [RFC5194], Section 7.1.

Since [RFC3428] and [RFC4975] are both based on SIP, ED-75 does not apply to all WebRTC-based emergency applications and services. As noted in "Emergency Services Functionality with the Extensible Messaging and Presence Protocol (XMPP)" [I-D.tschofenig-ecrit-xmpp-es], XMPP [RFC6120] is a potential alternative for emergency services

applications looking to support instant messaging [RFC6121] and multi-user chat [XEP-045] functionality.

"RTP Payload for Text Conversation" [RFC4103] is typically implemented along with SIP signaling as described in "Framework for Real-Time Text over IP Using the Session Initiation Protocol (SIP)" [RFC5194]. As a result, ED-76 does not apply to WebRTC implementations.

Alternatives to support of real-time text functionality are available, such as "In-Band Real Time Text" [XEP-0301], which supports real-time text by addition of child elements within XMPP message stanzas. The use of child elements to encapsulate real-time text, as well as transmission of complete lines enables [XEP-0301] to provide backward compatibility with existing XMPP instant-messaging and Multi-User Chat (MUC) clients, with no changes required to XMPP servers. Since XMPP can be encapsulated within HTTP via mechanisms such as BOSH [XEP-0206] or WebSockets [RFC6455], [XEP-0301] can be implemented in Javascript. Experience with Javascript implementation using the [Strophe] XMPP library indicates that adequate performance is achievable. In contrast, implementing real-time text as media as in [RFC4103] requires native browser support, as well as requiring changes to the configuration of intermediaries such as Session Border Controllers (SBCs). Also, [RFC4103] is not backward compatible with SIP instant messaging implementations supporting page-mode [RFC3428] or session [RFC4975] approaches.

## 5. Security Considerations

Security requirements in [RFC6881] include:

ED-48/SP-24 TLS [RFC5746] MUST be used to protect location (but see Section 9.1). All implementations MUST support TLS.

ED-58/SP-30 TLS is the primary mechanism used to secure the signaling for emergency calls. IPsec [RFC4301] MAY be used instead of TLS for any hop. Either TLS or IPSEC MUST be used when attempting to signal an emergency call.

ED-59/SP-31 If TLS session establishment is not available, or fails, the call MUST be retried without TLS.

ED-60/SP-32 [RFC5626] is RECOMMENDED to maintain persistent TLS connections between entities when one of the entity is an endpoint. Persistent TLS connection between proxies is RECOMMENDED using any suitable mechanism.

ED-61/AN-28 TLS SHOULD be used when attempting to retrieve

location (configuration or dereferencing) with HELD. The use of [RFC5077] is RECOMMENDED to minimize the time to establish TLS sessions without keeping server-side state. IPsec MAY be used instead of TLS.

ED-62/AN-29 When TLS session establishment fails, the location retrieval MUST be retried without TLS.

For WebRTC, HTTPS MUST be used to protect signaling for an emergency call, with potential fail-over to HTTP. HTTPS SHOULD be used to protect location retrieval (HELD) and call routing (LoST).

WebRTC security considerations are discussed in "Security Considerations for RTC-Web" [I-D.ietf-rtcweb-security]. The WebRTC security architecture, described in "RTCWEB Security Architecture" [I-D.ietf-rtcweb-security-arch], requires implementation of Secure RTP [RFC3711] as well as DTLS/SRTP [RFC5764].

While the security features of WebRTC exceed the requirements outlined in [RFC6881], support for emergency services within WebRTC raises concerns about potential attacks on the emergency services infrastructure, given the potential for malicious code to be executed within the browser. One way to lessen the likelihood of attacks by untrusted Javascript applications is for PSAPs to put up their own sites for emergency calling, protected by HTTPS.

While ICE [RFC5245] provides demonstration of liveness and consent to receive, it is possible for an attacker to overwhelm the PSAP by generating a large number of prank calls. IP relay services are also potential targets since these don't require forging of Caller-Id nor do they provide audio or video from the attacker.

Security threats to IP-based emergency services are described in "Security Threats and Requirements for Emergency Call Marking and Mapping" [RFC5069]. These include attacks on the emergency services system, such as attempting to deny system services to all users in a given area, to gain fraudulent use of services and to divert emergency calls to non-emergency sites. [RFC5069] also describes attacks against individuals, including attempts to prevent an individual from receiving aid, or to gain information about an emergency.

"Threat Analysis of the Geopriv Protocol" [RFC3694] describes threats against geographic location privacy, including protocol threats, threats resulting from the storage of geographic location data, and threats posed by the abuse of information.

Overall, experience indicates a relationship between anonymity and

the prevalence of prank calling. Therefore some protection may be provided through authentication of the caller either in the signaling or media plane. It is NOT RECOMMENDED that WebRTC-based emergency applications and services support anonymous emergency calling.

## 6. IANA Considerations

This document does not require actions by IANA.

## 7. Acknowledgments

We would like to thank the members of the IETF RTCWEB Working Group for discussions related to this topic.

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6881] Rosen, B. and J. Polk, "Best Current Practice for Communications Services in Support of Emergency Calling", RFC 6881, March 2013.
- [WCAG] Caldwell, B., Cooper, M., Reid, L.G. and G. Vanderheiden, "Web Content Accessibility Guidelines (WCAG) 2.0", <http://www.w3.org/TR/WCAG20/>, December 2008.

### 8.2. Informative References

- [GeolocationAPI] Popescu, A., "Geolocation API Specification", W3C, <http://dev.w3.org/geo/api/spec-source.html>
- [I.D.ietf-geopriv-res-gw-lis-discovery] Thomson, M. and R. Bellis, "Location Information Server (LIS) Discovery using IP address and Reverse DNS", draft-ietf-geopriv-res-gw-lis-discovery-05 (work in progress), April 2013.
- [I-D.ietf-rtcweb-overview] Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-06 (work in progress), February 2013.
- [I-D.ietf-rtcweb-rtp-usage] Perkins, C., Westerlund, M. and J. Ott, "Web Real-Time

Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-06 (work in progress), February 2013.

[I-D.ietf-rtcweb-security]

Rescorla, E., "Security Considerations for RTC-Web", draft-ietf-rtcweb-security-04 (work in progress), January 2013.

[I-D.ietf-rtcweb-security-arch]

Rescorla, E., "RTCWEB Security Architecture", draft-ietf-rtcweb-security-arch-06 (work in progress), July 2013.

[I-D.tschofenig-ecrit-xmpp-es]

Tschofenig, H., "Emergency Services Functionality with the Extensible Messaging and Presence Protocol (XMPP)", draft-tschofenig-ecrit-xmpp-es-00 (work in progress), March 2012.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3264] Rosenberg, J. and R. Schulzrinne, "An Offer/Answer Model with the Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3428] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3694] Danley, M., Mulligan, D., Morris, J. and J. Peterson, "Threat Analysis of the Geopriv Protocol", RFC 3694, February 2004.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E. and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4103] Hellstrom, G. and P. Jones, "RTP Payload for Text Conversation", RFC 4103, June 2005.

- [RFC4119] Peterson, J., "A Presence-based GEOPRIV Location Object Format", RFC 4119, December 2005.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC4975] Campbell, B., Mahy, R. and C. Jennings, "The Message Session Relay Protocol (MSRP)", RFC 4975, September 2007.
- [RFC5012] Schulzrinne, H. and R. Marshall, "Requirements for Emergency Context Resolution with Internet Technologies", RFC 5012, January 2008.
- [RFC5069] Taylor, T., Tschofenig, H., Schulzrinne, H. and M. Shanmugam, "Security Trheats and Requirements for Emergency Call Marking and Mapping", RFC 5069, January 2008.
- [RFC5077] Salowey, J., Zhou, H., Eronen, P. and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.
- [RFC5139] Thomson, M. and J. Winterbottom, "Revised Civic Location Format for Presence Information Data Format Location Object (PIDF-LO)", RFC 5139, February 2008.
- [RFC5194] van Wijk, A. and G. Gybels, "Framework for Real-Time Text over IP Using the Session Initiation Protocol (SIP)", RFC 5194, June 2008.
- [RFC5222] Hardie, T., Newton, A., Schulzrinne, H. and H. Tschofenig, "LoST: A Location-to-Service Translation Protocol", RFC 5222, August 2008.
- [RFC5223] Schulzrinne, H., Polk, J. and H. Tschofenig, "Discovering Location-to-Service Translation (LoST) Servers Using the Dynamic Host Configuration Protocol (DHCP)", RFC 5223, August 2008.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT", RFC 5389, October 2008.
- [RFC5626] Jennings, C., Mahy, R. and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.

- [RFC5746] Rescorla, E., Ray, M., Dispensa, S. and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, February 2010.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.
- [RFC5985] Barnes, M., "HTTP Enabled Location Delivery (HELD)", RFC 5985, September 2010.
- [RFC5986] Thomson, M. and J. Winterbottom, "Discovering the Local Location Information Server (LIS)", RFC 5986, September 2010.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6121] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", RFC 6121, March 2011.
- [RFC6184] Wang, Y.-K., Even, R., Kristensen, T. and R. Jesup, "RTP Payload Format for H.264 Video", RFC 6184, May 2011.
- [RFC6442] Polk, J., Rosen, B. and J. Peterson, "Location Conveyance for the Session Initiation Protocol", RFC 6442, December 2011.
- [RFC6443] Rosen, B., Schulzrinne, H., Polk, J. and A. Newton, "Framework for Emergency Calling Using Internet Multimedia", RFC 6443, December 2011.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- [StateMLTS] "State E911 Legislation",  
<http://www1.911enable.com/resource-center/state-e911-legislation>
- [Strophe] "Libraries for XMPP Poets", <http://strophe.im>
- [TTWG] "Proposed Timed Text Working Group Charter",  
<http://www.w3.org/2012/02/timed-text-wg-charter.html>
- [WEBRTC] Bergkvist, A., Burnett, D., Jennings, C. and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", W3C Editor's Draft (work in progress),

<http://dev.w3.org/2011/webrtc/editor/webrtc.html>, June 2013.

[WebTel] WebAPI/WebTelephony,  
<https://wiki.mozilla.org/WebAPI/WebTelephony>

[XEP-0206] Paterson, I. and P. Saint-Andre, "XMPP Over BOSH", XEP-0206 version 1.3, <http://xmpp.org/extensions/xep-0206.html>, July 2010.

[XEP-0301] Rejhon, M., "In-Band Real Time Text", XEP-0301 version 0.2, <http://xmpp.org/extensions/xep-0301.html>, March 2012.

[XEP-045] Saint-Andre, P., "Multi-User Chat", XEP 0045 version 1.25, <http://xmpp.org/extensions/xep-0045.html>, February 2012.

#### Authors' Addresses

Bernard Aboba  
Skype  
Redmond, WA 98052  
US

EMail: [bernard\\_aboba@hotmail.com](mailto:bernard_aboba@hotmail.com)

Martin Thomson  
Skype  
3210 Porter Drive  
Palo Alto, CA 94304  
US

Phone: +1 650-353-1925  
Email: [martin.thomson@gmail.com](mailto:martin.thomson@gmail.com)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: November 1, 2012

H. Alvestrand  
Google  
April 30, 2012

RTCWEB Resolution Negotiation  
draft-alvestrand-rtcweb-resolution-00

Abstract

This draft offers a proposal for a fragment of the SDP usage rules for RTCWEB: Requirements for supporting resolution negotiation.

It proposes to use SDP both for initial and within-call resolution configuration, and suggests that COP should be mentioned as an optional, not mandatory, mechanism.

Requirements Language

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

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 <http://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 November 1, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

## Table of Contents

1. Introduction . . . . .	3
2. Requirements . . . . .	3
3. Initial negotiation of parameters . . . . .	4
4. Per-stream declaration of desired video resolution . . . . .	5
4.1. SDP-based per-stream declaration . . . . .	5
4.2. COP-based per-stream declaration . . . . .	6
4.3. Tradeoffs discussion . . . . .	6
5. Usage considerations . . . . .	7
6. Relation to WebRTC API constraints . . . . .	8
7. IANA Considerations . . . . .	9
8. Security Considerations . . . . .	9
9. Acknowledgements . . . . .	9
10. References . . . . .	9
10.1. Normative References . . . . .	9
10.2. Informative References . . . . .	10
Author's Address . . . . .	10

## 1. Introduction

This draft offers a proposal for a fragment of the SDP usage rules for RTCWEB: Requirements for supporting resolution negotiation.

It proposes to use SDP, [RFC6236] in particular, both for initial and within-call resolution configuration, with the "a=recv-ssrc:imageattr" mechanism from [I-D.lennox-mmusic-sdp-source-selection] as a per-stream mechanism, and suggests that Codec Operation Point (COP) specified in [I-D.westerlund-avtext-codec-operation-point] should be mentioned as an optional, not mandatory, mechanism.

## 2. Requirements

The relevant requirement for video resolution negotiation from the RTCWEB use cases document [I-D.ietf-rtcweb-use-cases-and-requirements] is:

- o F25 The browser SHOULD use encoding of streams suitable for the current rendering (e.g. video display size) and SHOULD change parameters if the rendering changes during the session.

The scenarios from which this requirement is derived are:

- o 4.2.1 Simple Video Communication Service - user changes size of video during the session.
- o 4.2.2 Simple Video Communication Service, NAT/FW that blocks UDP - as above
- o 4.2.3 Simple Video Communication Service, global service provider - as above
- o 4.2.4 Simple Video Communication Service, enterprise aspects - as above
- o 4.2.5 Simple Video Communication Service, access change - bandwidth available changes dramatically during call (wired Ethernet to 3G)
- o 4.2.6 Simple Video Communication Service, QoS - as 4.2.5
- o 4.2.7 Simple Video Communication Service with sharing - as above
- o 4.2.8 Simple video communication service with inter-operator calling - as above

- o 4.2.10 Multiparty video communication - user changes size of video
- o (4.3.3 Video conferencing system with central server does NOT list F25 as a derived requirement, but notes that "it is important that the delay from when a video stream is selected for display until the video can be displayed is short").

Formulating the requirements in a form more amenable to implementation, there needs to be specified functions that allow the implementation:

- o To negotiate a maximum spatial resolution for all videos at call setup time
- o To negotiate a maximum temporal resolution ("frame rate") across all videos at call setup time
- o To negotiate other parameters as needed to ensure that the sender will not send a stream that the receiver is unable to handle.
- o To indicate the desire of the recipient for a particular spatial or temporal resolution on a particular video source, at any given time during the call
- o To indicate the intent of the sender to send a video source in a particular spatial or temporal resolution, at any given time during the call

This document does not mention other requirements.

### 3. Initial negotiation of parameters

We assume that the normal (payload-dependent) procedures for codec negotiation are sufficient to negotiate any codec parameters needed to ensure that the decoder can handle all incoming streams.

After the initial negotiation, the following variables MUST have a known value for each RTP session (represented by one or more m= lines):

- o The maximum X-resolution of any handlable video stream
- o The maximum Y-resolution of any handlable video stream
- o The maximum bitrate in bits per second

- o The maximum framerate in frames per second

An RTCWEB client MUST support negotiation of resolution using the "imageattr" attribute, as documented in [RFC6236].

An RTCWEB client MUST support a SAR value of 1.0 (square pixels), and MAY choose to support only the 1.0 value of the "sar" attribute.

The interpretation of the negotiation is that any video stream in the m= line containing the a=imageattr attribute will have a resolution within the bounds established by the negotiation.

An RTCWEB client MUST support negotiation of the "a=framerate" attribute, as specified in [RFC4566] section 6. Note that this is an upper bound on framerate; there is no lower bound negotiated.

These bounds MAY be renegotiated over the course of the call, but MUST NOT be renegotiated to render any currently transmitted video stream out of bounds

These bounds may be supplemented by payload-specific mechanisms, and there is no guarantee that all resolutions within the bounds can be supported.

#### 4. Per-stream declaration of desired video resolution

##### 4.1. SDP-based per-stream declaration

An RTCWEB client MUST support per-SSRC requests for video resolutions, as described in [I-D.lennox-mmusic-sdp-source-selection]. To be precise, it MUST support the a=remote-ssrc:<ssrc> framerate: and a=remote-ssrc:<ssrc>imageattr: attributes.

This satisfies the requirement to indicate the desire of the recipient for a particular spatial or temporal resolution.

We assume that the media sent from a sender to a receiver contains enough information inside the media format to tell what the resolution and framerate is.

The bounds specified for a single stream MUST be within the bounds previously negotiated for the whole session.

This mechanism does not form a negotiation; as specified in the referenced document, it is a declaration by the recipient of what stream formats he desires, and the sender will respond by changing

the video he sends. The sender SHOULD honor the requests by the receiver.

The mere fact that a stream is within the bounds negotiated for the session is not a sufficient condition for guaranteeing that the stream will be accepted; any number of issues, including temporary lack of resources at the recipient. Thus, the sender MUST always be prepared for one or more media streams to be refused by the recipient.

#### 4.2. COP-based per-stream declaration

An RTCWEB client MAY support the COP mechanism [I-D.westerlund-avtext-codec-operation-point] to negotiate the resolution of video within the limits established by the SDP negotiation without the need for additional SDP exchanges.

#### 4.3. Tradeoffs discussion

This section may be deleted before publication as an RFC; its main purpose is to discuss the decision to make the SDP-based mechanism a MUST and the COP-based mechanism a MAY.

Both mechanisms work by having the receiver declare a wish for a resolution, and the sender switching to that resolution. The main differences are:

- o In SDP, given the nature of the RTCWEB signalling model, the notification that a change is needed must be sent to the Javascript, which then has to use the createOffer mechanism to create a suitable SDP object, and use whatever mechanism is used for negotiation to send that request to the sender.
- o In COP, the decision to signal can possibly be taken either at Javascript level or inside the browser, but once the decision is taken, all further messaging is done by the browser using RTCP packets; Javascript is not involved.
- o For SDP, signalling follows the signalling path, which may be via a data channel along the media path, or may be via a completely different mechanism.
- o For COP, signalling always follows the media path's return path.
- o For SDP, the unbounded nature of the imageattr= specification allows a wide variety of sizes to be requested, including possibly unsuitable ones.

- o For COP, the list of alternatives is created explicitly using the Operation Point mechanism.
- o For SDP, the signalling transport is (presumably) done using a reliable transport
- o For COP, timeout and retransmission must be implemented in the requester.
- o For SDP, if imageattr= is already supported, the changes to the parsers involved are small.
- o For COP, support involves embedding a completely new functionality set within the RTCP components of the RTP-supporting libraries.
- o For SDP, the defining draft specifies some other mechanisms that are not mentioned here, such as "pause".
- o For COP, the defining draft specifies some configurations that are not part of the RTCWEB requirements set, such as multicast.
- o SDP does not consider the case of substreams for scalable video media.
- o COP does consider configuration of substreams.
- o For SDP, an IPR disclosure seeming to assert RF licensing has been made against the defining draft [ipr-ssrc].
- o For COP, an IPR disclosure asserting RAND (not RF) licensing has been made against the defining draft, with no assertion on which parts of the draft it applies to. [ipr-cop]

## 5. Usage considerations

This section notes briefly some of the situations in which resizing might be desirable.

- o Change of display window size on screen (window manager resize, for instance)
- o Changing the display target between a smaller and a larger window ("large current talker", for instance)
- o Retargeting of the display to a different display surface ("attach external monitor", for instance)

- o Temporary CPU or GPU overload due to media stream processing conflicting with other tasks, including handling a large number of media streams
- o Recovery from such overload situations
- o <<< more? >>>

Adaptation to bandwidth changes (congestion control) is NOT included in this set, since a more correct model for this is that it should be detected by the sender and the receiver operating in tandem, and the sender should decide which flows, if any, need their bitrates changed.

Turning video streams off (mute) is also not included; use of "size = 0" has been suggested as one mechanism for video mute, but this proposed mechanism is not addressed in this memo.

## 6. Relation to WebRTC API constraints

It is intended that the resolution negotiation be influenced by the constraints set by the application of either mandatory or optional constraints at the WebRTC API, as registered in the registry established by [I-D.burnett-rtcweb-constraints-registry].

The following relationships hold for all attributes that the implementation intends to satisfy (note that the constraints listed here have NOT been registered yet):

video-min-height >= value of imageattr y= xyrange lower bound

video-max-height <= value of imageattr y= xyrange upper bound

video-min-framerate is not represented in SDP

video-max-framerate <= value of a=framerate attribute

video-min-aspect-ratio <= value of imageattr "par=" prange lower bound

video-max-aspect-ratio >= value of imageattr "par=" prange upper bound

The implementation is free to increase "min" values or decrease "max" values (make requirements more restrictive) and add "step" in order to fit with its implementation restrictions.

Constraints specified at PeerConnection creation time are reflected as SDP-wide values. Constraints specified when creating a MediaStream or attaching a MediaStream to a PeerConnection are reflected as ssrc-specific values.

The envisioned usage is that the application will not use the values specified by the client directly, but choose the minimum of the specified bounds and the implementation limitations of the browser, adjusted for any odd requirements of the codec or soft/hardware, and choose a representation in the SDP that adequately represents the possible configurations.

## 7. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 8. Security Considerations

All considerations related to normal usage of SDP apply to this memo.

## 9. Acknowledgements

## 10. References

### 10.1. Normative References

[I-D.burnett-rtcweb-constraints-registry]

Burnett, D., "IANA Registry for RTCWeb Media Constraints", draft-burnett-rtcweb-constraints-registry-00 (work in progress), March 2012.

[I-D.ietf-rtcweb-use-cases-and-requirements]

Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-06 (work in progress), October 2011.

[I-D.lennox-mmusic-sdp-source-selection]

Lennox, J. and H. Schulzrinne, "Mechanisms for Media Source Selection in the Session Description Protocol (SDP)", draft-lennox-mmusic-sdp-source-selection-03 (work

in progress), January 2012.

- [I-D.westerlund-avtext-codec-operation-point]  
Westerlund, M., Burman, B., and L. Hamm, "Codec Operation Point RTCP Extension",  
draft-westerlund-avtext-codec-operation-point-00 (work in progress), March 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.
- [RFC6236] Johansson, I. and K. Jung, "Negotiation of Generic Image Attributes in the Session Description Protocol (SDP)", RFC 6236, May 2011.

#### 10.2. Informative References

- [ipr-cop] "Telefonaktiebolaget LM Ericsson (publ)'s Statement about IPR related to  
draft-westerlund-avtext-codec-operation-point-00 -  
<https://datatracker.ietf.org/ipr/1701/>", March 2012.
- [ipr-ssrc] "Vidyo, Inc.'s Statement about IPR related to  
draft-lennox-mmusic-sdp-source-selection-00 -  
<https://datatracker.ietf.org/ipr/1170/>".

#### Author's Address

Harald Alvestrand  
Google

Email: [harald@alvestrand.no](mailto:harald@alvestrand.no)



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 13, 2012

C. Bran  
Plantronics  
C. Jennings  
Cisco  
JM. Valin  
Mozilla  
March 12, 2012

WebRTC Codec and Media Processing Requirements  
draft-cbran-rtcweb-codec-02

Abstract

This document outlines the codec and media processing requirements for WebRTC client application and endpoint devices.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

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 <http://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 September 13, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	3
3. Codec Requirements . . . . .	3
3.1. Audio Codec Requirements . . . . .	3
3.2. Video Codec Requirements . . . . .	3
4. Audio Level . . . . .	4
5. Acoustic Echo Cancellation (AEC) . . . . .	5
6. Legacy VoIP Interoperability . . . . .	6
7. IANA Considerations . . . . .	6
8. Security Considerations . . . . .	6
9. Acknowledgements . . . . .	6
10. Normative References . . . . .	6
Authors' Addresses . . . . .	7

## 1. Introduction

An integral part of the success and adoption of the Web Real Time Communications (WebRTC) will be the voice and video interoperability between WebRTC applications. This specification will outline the media processing and codec requirements for WebRTC client implementations.

## 2. Terminology

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

## 3. Codec Requirements

This section covers the audio and video codec requirements for WebRTC client applications. To ensure a baseline level of interoperability between WebRTC clients, a minimum set of required codecs are specified below. While this section specifies the codecs that will be mandated for all WebRTC client implementations, it leaves the question of supporting additional codecs to the will of the implementer.

### 3.1. Audio Codec Requirements

WebRTC clients are REQUIRED to implement the following audio codecs.

- o PCMA/PCMU - 1 channel with a rate of 8000 Hz and a ptime of 20 - see section 4.5.14 of [RFC3551]
- o Telephone Event - [RFC4734]
- o Opus [draft-ietf-codec-opus]

For all cases where the client is able to process audio at a sampling rate higher than 8 kHz, it is RECOMMENDED that Opus be offered before PCMA/PCMU. For Opus, all modes MUST be supported, for all ptime values up to 120 ms. Clients MAY use the offer/answer mechanism to signal a preference for a particular mode or ptime.

### 3.2. Video Codec Requirements

The following feature list applies to all required video codecs.

Required video codecs:

- o MUST support at least 10 frames per second (fps) and SHOULD support 30 fps
- o If VP8 is supported, then it MUST support the bilinear and none reconstruction filters
- o OPTIONALLY offer support for additional color spaces
- o MUST support a minimum resolution of 320X240
- o SHOULD support resolutions of 1280x720, 720x480, 1024x768, 800x600, 640x480, 640 x 360 , 320x240

#### 4. Audio Level

It is desirable to standardize the "on the wire" audio level for speech transmission to avoid users having to manually adjust the playback and to facilitate mixing in conferencing applications. It is also desirable to be consistent with ITU-T recommendations G.169 and G.115, which recommend an active audio level of -19 dBm0. However, unlike G.169 and G.115, the audio for WebRTC is not constrained to have a passband specified by G.712 and can in fact be sampled at any sampling rate from 8 kHz to 48 kHz and up. For this reason, the level SHOULD be normalized by only considering frequencies above 300 Hz, regardless of the sampling rate used. The level SHOULD also be adapted to avoid clipping, either by lowering the gain to a level below -19 dBm0, or through the use of a compressor.

AUTHORS' NOTE: The idea of using the same level as what the ITU-T recommends is that it should improve inter-operability while at the same time maintaining sufficient dynamic range and reducing the risk of clipping. The main drawbacks are that the resulting level is about 12 dB lower than typical "commercial music" levels and it leaves room for ill-behaved clients to be much louder than a normal client. While using music-type levels is not really an option (it would require using the same compressor-limitors that studios use), it would be possible to have a level slightly higher (e.g. 3 dB) than what is recommended above without causing interoperability problems.

Assuming 16-bit PCM with a value of +/-32767, -19 dBm0 corresponds to a root mean square (RMS) level of 2600. Only active speech should be considered in the RMS calculation. If the client has control over the entire audio capture path, as is typically the case for a regular phone, then it is RECOMMENDED that the gain be adjusted in such a way that active speech have a level of 2600 (-19 dBm0) for an average speaker. If the client does not have control over the entire audio

capture, as is typically the case for a software client, then the client SHOULD use automatic gain control (AGC) to dynamically adjust the level to 2600 (-19 dBm0) +/- 6 dB. For music or desktop sharing applications, the level SHOULD NOT be automatically adjusted and the client SHOULD allow the user to set the gain manually.

The RECOMMENDED filter for normalizing the signal energy is a second-order Butterworth filter with a 300 Hz cutoff frequency.

It is common for the audio output on some devices to be "calibrated" for playing back pre-recorded "commercial" music, which is typically around 12 dB louder than the level recommended in this section. Because of this, clients MAY increase the gain before playback.

## 5. Acoustic Echo Cancellation (AEC)

It is plausible that the dominant near to mid-term WebRTC usage model will be people using the interactive audio and video capabilities to communicate with each other via web browsers running on a notebook computer that has built-in microphone and speakers. The notebook-as-communication-device paradigm presents challenging echo cancellation problems, the specific remedy of which will not be mandated here. However, while no specific algorithm or standard will be required by WebRTC compatible clients, echo cancellation will improve the user experience and should be implemented by the endpoint device.

SHOULD include an AEC and if not, SHOULD ensure that the speaker-to-microphone gain is below unity at all frequencies to avoid instability when none of the client has echo cancellation. For clients that do not control the audio capture and playback devices directly, it is RECOMMENDED to support echo cancellation between devices running at slight different sampling rates, such as when a webcam is used for microphone.

The client SHOULD allow either the entire AEC or the non-linear processing (NLP) to be turned off for applications, such as music, that do not behave well with the spectral attenuation methods typically used in NLPs. It SHOULD have the ability to detect the presence of a headset and disable echo cancellation.

For some applications where the remote client may not have an echo canceller, the local client MAY include a far-end echo canceller, but if that is the case, it SHOULD be disabled by default.

Call control event notification to connected devices such as headsets (what's that exactly?)

## 6. Legacy VoIP Interoperability

The codec requirements above will ensure, at a minimum, voice interoperability capabilities between WebRTC client applications and legacy phone systems.

## 7. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 8. Security Considerations

The codec requirements have no additional security considerations other than those captured in [I-D.ekr-security-considerations-for-rtc-web].

## 9. Acknowledgements

This draft incorporates ideas and text from various other drafts. In particular we would like to acknowledge, and say thanks for, work we incorporated from Harald Alvestrand.

## 10. Normative References

- [I-D.ekr-security-considerations-for-rtc-web]  
Rescorla, E., "Security Considerations for RTC-Web",  
May 2011.
- [I-D.webm]  
Google, Inc., "VP8 Data Format and Decoding Guide",  
July 2010.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and  
Video Conferences with Minimal Control", STD 65, RFC 3551,  
July 2003.
- [RFC4734] Schulzrinne, H. and T. Taylor, "Definition of Events for  
Modem, Fax, and Text Telephony Signals", RFC 4734,

December 2006.

Authors' Addresses

Cary Bran  
Plantronics  
345 Encinial Street  
Santa Cruz, CA 95060  
USA

Phone: +1 206 661-2398  
Email: cary.bran@plantronics.com

Cullen Jennings  
Cisco  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Phone: +1 408 421-9990  
Email: fluffy@cisco.com

Jean-Marc Valin  
Mozilla  
650 Castro Street  
Mountain View, CA 94041  
USA

Email: jmvalin@jmvalin.ca



AVTCORE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: April 23, 2015

J. Mattsson, Ed.  
Ericsson  
D. McGrew  
D. Wing  
F. Andreassen  
Cisco  
October 20, 2014

Encrypted Key Transport for Secure RTP  
draft-ietf-avtcore-srtp-ekt-03

Abstract

Encrypted Key Transport (EKT) is an extension to Secure Real-time Transport Protocol (SRTP) that provides for the secure transport of SRTP master keys, Rollover Counters, and other information. This facility enables SRTP to work for decentralized conferences with minimal control.

This note defines EKT, and also describes how to use it with SDP Security Descriptions, DTLS-SRTP, and MIKEY. With EKT, these other key management protocols provide an EKT key to everyone in a session, and EKT coordinates the SRTP keys within the session.

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 <http://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 April 23, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	3
1.1. History . . . . .	4
1.2. Conventions Used In This Document . . . . .	5
2. Encrypted Key Transport . . . . .	5
2.1. EKT Field Formats . . . . .	6
2.2. Packet Processing and State Machine . . . . .	8
2.2.1. Outbound Processing . . . . .	8
2.2.2. Inbound Processing . . . . .	9
2.3. Ciphers . . . . .	11
2.3.1. The Default Cipher . . . . .	12
2.3.2. Other EKT Ciphers . . . . .	13
2.4. Synchronizing Operation . . . . .	13
2.5. Transport . . . . .	13
2.6. Timing and Reliability Consideration . . . . .	15
3. Use of EKT with SDP Security Descriptions . . . . .	16
3.1. SDP Security Descriptions Recap . . . . .	16
3.2. Relationship between EKT and SDESC . . . . .	17
3.3. Overview of Combined EKT and SDESC Operation . . . . .	19
3.4. EKT Extensions to SDP Security Descriptions . . . . .	19
3.5. Offer/Answer Considerations . . . . .	20
3.5.1. Generating the Initial Offer - Unicast Streams . . . . .	20
3.5.2. Generating the Initial Answer - Unicast Streams . . . . .	21
3.5.3. Processing of the Initial Answer - Unicast Streams . . . . .	22
3.6. SRTP-Specific Use Outside Offer/Answer . . . . .	23
3.7. Modifying the Session . . . . .	23
3.8. Backwards Compatibility Considerations . . . . .	24
3.9. Grammar . . . . .	25
4. Use of EKT with DTLS-SRTP . . . . .	25
4.1. DTLS-SRTP Recap . . . . .	26
4.2. EKT Extensions to DTLS-SRTP . . . . .	26
4.3. Offer/Answer Considerations . . . . .	28
4.3.1. Generating the Initial Offer . . . . .	28
4.3.2. Generating the Initial Answer . . . . .	29
4.3.3. Processing the Initial Answer . . . . .	29
4.3.4. Sending DTLS EKT Key Reliably . . . . .	30
4.3.5. Modifying the Session . . . . .	30

5. Use of EKT with MIKEY . . . . .	30
5.1. EKT Extensions to MIKEY . . . . .	32
5.2. Offer/Answer Considerations . . . . .	33
5.2.1. Generating the Initial Offer . . . . .	33
5.2.2. Generating the Initial Answer . . . . .	34
5.2.3. Processing the Initial Answer . . . . .	34
5.2.4. Modifying the Session . . . . .	35
6. Using EKT for Interoperability between Key Management Systems	35
7. Design Rationale . . . . .	36
7.1. Alternatives . . . . .	37
8. Security Considerations . . . . .	37
9. IANA Considerations . . . . .	39
10. Acknowledgements . . . . .	39
11. References . . . . .	40
11.1. Normative References . . . . .	40
11.2. Informative References . . . . .	41
Appendix A. Using EKT to Optimize Interworking DTLS-SRTP with Security Descriptions . . . . .	42
Authors' Addresses . . . . .	44

## 1. Introduction

RTP is designed to allow decentralized groups with minimal control to establish sessions, such as for multimedia conferences. Unfortunately, Secure RTP (SRTP [RFC3711]) cannot be used in many minimal-control scenarios, because it requires that SSRC values and other data be coordinated among all of the participants in a session. For example, if a participant joins a session that is already in progress, that participant needs to be told the SRTP keys (and SSRC, ROC and other details) of the other SRTP sources.

The inability of SRTP to work in the absence of central control was well understood during the design of the protocol; the omission was considered less important than optimizations such as bandwidth conservation. Additionally, in many situations SRTP is used in conjunction with a signaling system that can provide most of the central control needed by SRTP. However, there are several cases in which conventional signaling systems cannot easily provide all of the coordination required. It is also desirable to eliminate the layer violations that occur when signaling systems coordinate certain SRTP parameters, such as SSRC values and ROCs.

This document defines Encrypted Key Transport (EKT) for SRTP, an extension to SRTP that fits within the SRTP framework and reduces the amount of external signaling control that is needed in an SRTP session. EKT securely distributes the SRTP master key and other information for each SRTP source (SSRC), using SRTCP or SRTP to transport that information. With this method, SRTP entities are free

to choose SSRC values as they see fit, and to start up new SRTP sources (SSRC) with new SRTP master keys (see Section 2.2) within a session without coordinating with other entities via external signaling or other external means. This fact allows to reinstate the RTP collision detection and repair mechanism, which is nullified by the current SRTP specification because of the need to control SSRC values closely. An SRTP endpoint using EKT can generate new keys whenever an existing SRTP master key has been overused, or start up a new SRTP source (SSRC) to replace an old SRTP source that has reached the packet-count limit. However, EKT does not allow SRTP's ROC to rollover; that requires re-keying outside of EKT (e.g., using MIKEY or DTLS-SRTP). EKT also solves the problem in which the burst loss of the N initial SRTP packets can confuse an SRTP receiver, when the initial RTP sequence number is greater than or equal to  $2^{16} - N$ . These features can simplify many architectures that implement SRTP.

EKT provides a way for an SRTP session participant, either a sender or receiver, to securely transport its SRTP master key and current SRTP rollover counter to the other participants in the session. This data, possibly in conjunction with additional data provided by an external signaling protocol, furnishes the information needed by the receiver to instantiate an SRTP/SRTCP receiver context.

EKT does not control the manner in which the SSRC is generated; it is only concerned with their secure transport. Those values may be generated on demand by the SRTP endpoint, or may be dictated by an external mechanism such as a signaling agent or a secure group controller.

EKT is not intended to replace external key establishment mechanisms such as SDP Security Descriptions [RFC4568], DTLS-SRTP [RFC5764], or MIKEY [RFC3830][RFC4563]. Instead, it is used in conjunction with those methods, and it relieves them of the burden of tightly coordinating every SRTP source (SSRC) among every SRTP participant.

### 1.1. History

[[RFC Editor Note: please remove this section prior to publication as an RFC.]]

A substantial change occurred between the EKT documents draft-ietf-avt-srtp-ekt-03 and draft-ietf-avtcore-srtp-ekt-00. The change makes it possible for the EKT data to be removed from a packet without affecting the ability of the receiver to correctly process the data that is present in that packet. This capability facilitates interoperability between SRTP implementations with different SRTP key management methods. The changes also greatly simplify the EKT

processing rules, and makes the EKT data that must be carried in SRTP and/or SRTCP packets somewhat larger.

In draft-ietf-avtcore-srtp-ekt-02, SRTP master keys have to be always generated randomly and not re-used, MKI is no longer allowed with EKT (as MKI duplicates some of EKT's functions), and text clarifies that EKT must be negotiated during call setup. Some text was consolidated and re-written, notably Section 2.6 ("Timing and Reliability"). Support for re-directing the DTLS-SRTP handshake to another host was removed, as it needed NAT traversal support.

In draft-ietf-avtcore-srtp-ekt-03, the SRTCP compound packet problem is discussed. Updates and clarifications were made to the SDESC and MIKEY sections.

## 1.2. Conventions Used In This Document

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

## 2. Encrypted Key Transport

In EKT, an SRTP master key is encrypted with a key encrypting key and the resulting ciphertext is transported in selected SRTCP packets or in selected SRTP packets. The key encrypting key is called an EKT key. A single such key suffices for a single SRTP session, regardless of the number of participants in that session. However, there can be multiple EKT keys used within a particular session.

EKT defines a new method of providing SRTP master keys to an endpoint. In order to convey the ciphertext of the SRTP master key, and other additional information, an additional EKT field is added to SRTP or SRTCP packets. When added to SRTCP, the EKT field appears at the end of the packet, after the authentication tag, if that tag is present, or after the SRTCP index otherwise. When added to SRTP, The EKT field appears at the end of the SRTP packet, after the authentication tag (if that tag is present), or after the ciphertext of the encrypted portion of the packet otherwise.

EKT MUST NOT be used in conjunction with SRTP's MKI (Master Key Identifier) or with SRTP's <From, To> [RFC3711], as those SRTP features duplicate some of the functions of EKT.

## 2.1. EKT Field Formats

The EKT Field uses one of the two formats defined below. These two formats can always be unambiguously distinguished on receipt by examining the final bit of the EKT Field, which is also the final bit of the SRTP or SRTCP packet. The first format is the Full EKT Field (or Full\_EKT\_Field), and the second is the Short EKT Field (or Short\_EKT\_Field). The formats are defined as

```
EKT_Plaintext = SRTP_Master_Key || SSRC || ROC || ISN  
  
EKT_Ciphertext = EKT_Encrypt(EKT_Key, EKT_Plaintext)  
  
Full_EKT_Field = EKT_Ciphertext || SPI || '1'  
  
Short_EKT_Field = Reserved || '0'
```

Figure 1: EKT data formats

Here || denotes concatenation, and '1' and '0' denote single one and zero bits, respectively. These fields and data elements are defined as follows:

**EKT\_Plaintext:** The data that is input to the EKT encryption operation. This data never appears on the wire, and is used only in computations internal to EKT.

**EKT\_Ciphertext:** The data that is output from the EKT encryption operation, described in Section 2.3. This field is included in SRTP and SRTCP packets when EKT is in use. The length of this field is variable, and is equal to the ciphertext size N defined in Section 2.3. Note that the length of the field is inferable from the SPI field, since the particular EKT cipher used by the sender of a packet can be inferred from that field.

**SRTP\_Master\_Key:** On the sender side, the SRTP Master Key associated with the indicated SSRC. The length of this field depends on the cipher suite negotiated during call setup for SRTP or SRTCP.

**SSRC:** On the sender side, this field is the SSRC for this SRTP source. The length of this field is fixed at 32 bits.

**Rollover Counter (ROC):** On the sender side, this field is set to the current value of the SRTP rollover counter in the SRTP context associated with the SSRC in the SRTP or SRTCP packet. The length of this field is fixed at 32 bits.

Initial Sequence Number (ISN): If this field is nonzero, it indicates the RTP sequence number of the initial RTP packet that is protected using the SRTP master key conveyed (in encrypted form) by the EKT Ciphertext field of this packet. When this field is present in an RTCP packet it indicates the RTP sequence number of the first RTP packet encrypted by this master key. If the ISN field is zero, it indicates that the initial RTP/RTCP packet protected using the SRTP master key conveyed in this packet preceded, or was concurrent with, the last roll-over of the RTP sequence number, and thus should be used as the current master key for processing this packet. The length of this field is fixed at 16 bits.

Security Parameter Index (SPI): This field is included in SRTP and SRTCP packets when EKT is in use. It indicates the appropriate EKT key and other parameters for the receiver to use when processing the packet. It is an "index" into a table of possibilities (which are established via signaling or some other out-of-band means), much like the IPsec Security Parameter Index [RFC4301]. The length of this field is fixed at 15 bits. The parameters identified by this field are:

- \* The EKT key used to process the packet.
- \* The EKT cipher used to process the packet.
- \* The Secure RTP parameters associated with the SRTP Master Key carried by the packet and the SSRC value in the packet. Section 8.2. of [RFC3711] summarizes the parameters defined by that specification.
- \* The Master Salt associated with the Master Key. (This value is part of the parameters mentioned above, but we call it out for emphasis.) The Master Salt is communicated separately, via signaling, typically along with the EKT key.

Together, these data elements are called an EKT parameter set. Within each SRTP session, each distinct EKT parameter set that may be used MUST be associated with a distinct SPI value, to avoid ambiguity.

Reserved: The length of this field is 7 bits. MUST be all zeros on transmission, and MUST be ignored on reception.

The Full\_EKT\_Field and Short\_EKT\_Field formats are shown in Figure 2 and Figure 3, respectively. These figures show the on-the-wire data. The Ciphertext field holds encrypted data, and thus has no apparent inner structure.

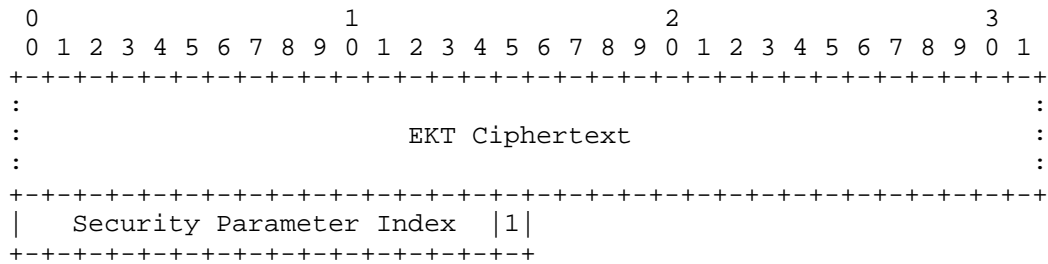


Figure 2: Full EKT Field format

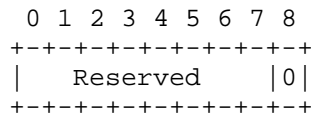


Figure 3: Short EKT Field format

## 2.2. Packet Processing and State Machine

At any given time, each SRTP/SRTCP source (SSRC) has associated with it a single EKT parameter set. This parameter set is used to process all outbound packets, and is called the outbound parameter set. There may be other EKT parameter sets that are used by other SRTP/SRTCP sources in the same session, including other SRTP/SRTCP sources on the same endpoint (e.g., one endpoint with voice and video might have two EKT parameter sets, or there might be multiple video sources on an endpoint each with their own EKT parameter set). All of these EKT parameter sets SHOULD be stored by all of the participants in an SRTP session, for use in processing inbound SRTP and SRTCP traffic.

All SRTP master keys MUST NOT be re-used, MUST be randomly generated according to [RFC4086], and MUST NOT be equal to or derived from other SRTP master keys.

### 2.2.1. Outbound Processing

See Section 2.6 which describes when to send an EKT packet and describes if a Full EKT Field or Short EKT Field is sent.

When an SRTP or SRTCP packet is to be sent, the EKT field for that packet is created as follows, or uses an equivalent set of steps. The creation of the EKT field MUST precede the normal SRTP or SRTCP packet processing. The ROC used in EKT processing MUST be the same as the one used in the SRTP processing.

If the Short format is used, an all-zero octet is appended to the packet. Otherwise, processing continues as follows.

The Rollover Counter field in the packet is set to the current value of the SRTP rollover counter (represented as an unsigned integer in network byte order).

The Initial Sequence Number field is set to zero, if the initial RTP packet protected using the current SRTP master key for this source preceded, or was concurrent with, the last roll-over of the RTP sequence number. Otherwise, that field is set to the value of the RTP sequence number of the initial RTP packet that was or will be protected by that key. See "rekey" in Section 2.6. The rekeying event MUST NOT change the value of ROC (otherwise, the current value of the ROC would not be known to late joiners of existing sessions). This means rekeying near the end of sequence number space (e.g., 100 packets before sequence number 65535) is not possible because ROC needs to roll over.

The Security Parameter Index field is set to the value of the Security Parameter Index that is associated with the outbound parameter set.

The EKT\_Plaintext field is computed from the SRTP Master Key, SSRC, ROC, and ISN fields, as shown in Figure 1.

The EKT\_Ciphertext field is set to the ciphertext created by encrypting the EKT\_Plaintext with the EKT cipher, using the EKT Key as the encryption key. The encryption process is detailed in Section 2.3. Implementations MAY cache the value of this field to avoid recomputing it for each packet that is sent.

Implementation note: Because of the format of the Full EKT Field, a packet containing the Full EKT Field MUST be sent when the ROC changes (i.e., every  $2^{16}$  packets).

#### 2.2.2. Inbound Processing

When an SRTP or SRTCP packet containing a Full EKT Field or Short EKT Field is received, it is processed as follows or using an equivalent set of steps. Inbound EKT processing MUST take place prior to the usual SRTP or SRTCP processing. Implementation note: the receiver may want to have a sliding window to retain old master keys for some brief period of time, so that out of order packets can be processed. The following steps show processing as packets are received in order.

1. The final bit is checked to determine which EKT format is in use. If the packet contains a Short EKT Field then the Short EKT Field

is removed and normal SRTP or SRTCP processing is applied. If the packet contains a Full EKT Field, then processing continues as described below.

2. The Security Parameter Index (SPI) field is checked to determine which EKT parameter set should be used when processing the packet. If multiple parameter sets have been defined for the SRTP session, then the one that is associated with the value of the SPI field in the packet is used. This parameter set is called the matching parameter set below. If there is no matching SPI, then the verification function MUST return an indication of authentication failure, and the steps described below are not performed.
3. The EKT\_Ciphertext is decrypted using the EKT\_Key and EKT\_Cipher in the matching parameter set, as described in Section 2.3. If the EKT decryption operation returns an authentication failure, then the packet processing halts with an indication of failure. Otherwise, the resulting EKT\_Plaintext is parsed as described in Figure 1, to recover the SRTP Master Key, SSRC, ROC, and ISN fields.
4. The SSRC field output from the decryption operation is compared to the SSRC field from the SRTP header if EKT was received over SRTP, or from the SRTCP header if EKT was received over SRTCP. If the values of the two fields do not match, then packet processing halts with an indication of failure. Otherwise, it continues as follows.
5. If an SRTP context associated with the SSRC in the previous step already exists and the ROC from the EKT\_Plaintext is less than the ROC in the SRTP context, then EKT processing halts and the packet is processed as an out-of-order packet (if within the implementation's sliding window) or dropped (as it is a replay). Otherwise, the ROC in the SRTP context is set to the value of the ROC from the EKT\_Plaintext, and the SRTP Master Key from the EKT\_Plaintext is accepted as the SRTP master key corresponding to the SSRC indicated in the EKT\_Plaintext, beginning at the sequence number indicated by the ISN (see next step).
6. If the ISN from the EKT\_Plaintext is less than the RTP sequence number of an authenticated received SRTP packet, then EKT processing halts (as this is a replay). If the Initial Sequence Number field is nonzero, then the initial sequence number for the SRTP master key is set to the packet index created by appending that field to the current rollover counter and treating the result as a 48-bit unsigned integer. The initial sequence number for the master key is equivalent to the "From" value of the

<From, To> pair of indices (Section 8.1.1 of [RFC3711]) that can be associated with a master key.

7. The newly accepted SRTP master key, the SRTP parameters from the matching parameter set, and the SSRC from the packet are stored in the crypto context associated with the SRTP source (SSRC). The SRTP Key Derivation algorithm is run in order to compute the SRTP encryption and authentication keys, and those keys are stored for use in SRTP processing of inbound packets. The Key Derivation algorithm takes as input the newly accepted SRTP master key, along with the Master Salt from the matching parameter set.
8. At this point, EKT processing has successfully completed, and the normal SRTP or SRTCP processing takes place.

Implementation note: the value of the EKT Ciphertext field is identical in successive packets protected by the same EKT parameter set and the same SRTP master key, ROC, and ISN. This ciphertext value MAY be cached by an SRTP receiver to minimize computational effort by noting when the SRTP master key is unchanged and avoiding repeating Steps 2 through 6.

### 2.3. Ciphers

EKT uses an authenticated cipher to encrypt the EKT Plaintext, which is comprised of the SRTP master keys, SSRC, ROC, and ISN. We first specify the interface to the cipher, in order to abstract the interface away from the details of that function. We then define the cipher that is used in EKT by default. The default cipher described in Section 2.3.1 MUST be implemented, but another cipher that conforms to this interface MAY be used, in which case its use MUST be coordinated by external means (e.g., key management).

The master salt length for the offered cipher suites MUST be the same. In practice the easiest way to achieve this is by offering the same crypto suite.

An EKT cipher consists of an encryption function and a decryption function. The encryption function  $E(K, P)$  takes the following inputs:

- o a secret key  $K$  with a length of  $L$  bytes, and
- o a plaintext value  $P$  with a length of  $M$  bytes.

The encryption function returns a ciphertext value  $C$  whose length is  $N$  bytes, where  $N$  is at least  $M$ . The decryption function  $D(K, C)$  takes the following inputs:

- o a secret key  $K$  with a length of  $L$  bytes, and
- o a ciphertext value  $C$  with a length of  $N$  bytes.

The decryption function returns a plaintext value  $P$  that is  $M$  bytes long, or returns an indication that the decryption operation failed because the ciphertext was invalid (i.e. it was not generated by the encryption of plaintext with the key  $K$ ).

These functions have the property that  $D(K, E(K, P)) = P$  for all values of  $K$  and  $P$ . Each cipher also has a limit  $T$  on the number of times that it can be used with any fixed key value. For each key, the encryption function **MUST NOT** be invoked on more than  $T$  distinct values of  $P$ , and the decryption function **MUST NOT** be invoked on more than  $T$  distinct values of  $C$ .

The length of the EKT Plaintext is ten bytes, plus the length of the SRTP Master Key.

Security requirements for EKT ciphers are discussed in Section 8.

#### 2.3.1. The Default Cipher

The default EKT Cipher is the Advanced Encryption Standard (AES) [FIPS197] Key Wrap with Padding [RFC5649] algorithm. It requires a plaintext length  $M$  that is at least one octet, and it returns a ciphertext with a length of  $N = M + 8$  octets. It can be used with key sizes of  $L = 16, 24$ , and  $32$ , and its use with those key sizes is indicated as AESKW\_128, AESKW\_192, and AESKW\_256, respectively. The key size determines the length of the AES key used by the Key Wrap algorithm. With this cipher,  $T=2^{48}$ .

SRTP transform	EKT transform	length of EKT plaintext	length of EKT ciphertext	length of Full EKT Field
AES-128	AESKW_128 (m)	26	40	42
AES-192	AESKW_192	34	48	50
AES-256	AESKW_256	42	56	58
F8-128	AESKW_128	26	40	42
SEED-128	AESKW_128	26	40	42

Figure 4: AESKW Table

The mandatory to implement transform is AESKW\_128, indicated by (m).

As AES-128 is the mandatory to implement transform in SRTP [RFC3711], AESKW\_128 MUST be implemented for EKT.

For all the SRTP transforms listed in the table, the corresponding EKT transform MUST be used, unless a stronger EKT transform is negotiated by key management.

#### 2.3.2. Other EKT Ciphers

Other specifications may extend this one by defining other EKT ciphers per Section 9. This section defines how those ciphers interact with this specification.

An EKT cipher determines how the EKT Ciphertext field is written, and how it is processed when it is read. This field is opaque to the other aspects of EKT processing. EKT ciphers are free to use this field in any way, but they SHOULD NOT use other EKT or SRTP fields as an input. The values of the parameters L, M, N, and T MUST be defined by each EKT cipher, and those values MUST be inferable from the EKT parameter set.

#### 2.4. Synchronizing Operation

A participant in a session MAY opt to use a particular EKT parameter set to protect outbound packets after it accepts that EKT parameter set for protecting inbound traffic. In this case, the fact that one participant has changed to using a new EKT key for outbound traffic can trigger other participants to switch to using the same key.

If a source has its EKT key changed by key management, it MUST also change its SRTP master key, which will cause it to send out a new Full EKT Field. This ensures that if key management thought the EKT key needs changing (due to a participant leaving or joining) and communicated that in key management to a source, the source will also change its SRTP master key, so that traffic can be decrypted only by those who know the current EKT key.

The use of EKT MUST be negotiated during key management or call setup (e.g., using DTLS-SRTP, Security Descriptions, MIKEY, or similar).

#### 2.5. Transport

EKT SHOULD be used over SRTP, and MAY be used over SRTCP. SRTP is preferred because it shares fate with transmitted media, because SRTP rekeying can occur without concern for RTCP transmission limits, and to avoid SRTCP compound packets with RTP translators and mixers.

This specification requires the EKT SSRC match the SSRC in the RTCP header, but Section 6.1 of [RFC3550] encourages creating SRTCP compound packets:

It is RECOMMENDED that translators and mixers combine individual RTCP packets from the multiple sources they are forwarding into one compound packet whenever feasible in order to amortize the packet overhead (see Section 7).

These compound SRTCP packets might have an SSRC that does not match the EKT SSRC. To reduce the occasion of this occurring, EKT-aware RTP mixers and translators which are generating SRTCP compound packets SHOULD attempt to place an SRTCP payload containing an EKT tag at the front of the compound packet (so that the EKT receiver will process it), and MAY be even more robust and implement more sophisticated algorithms to ensure all EKT tags from different senders are sent at the front of the compound packet. However, no robust algorithm exists which ensures robust EKT delivery in conjunction with SRTCP compound packets. This impact to RTP translators and mixers, and the inability to reliably determine an RTP translator or mixer might be involved in an RTP session, provides further incentive to send EKT over RTP.

The packet processing, state machine, and Authentication Tag format for EKT over SRTP are nearly identical to that for EKT over SRTCP. Differences are highlighted in Section 2.2.1 and Section 2.2.2.

The Full EKT Field is appended to the SRTP or SRTCP payload and is 42, 50, or 58 octets long for AES-128, AES-192, or AES-256, respectively. This length impacts the maximum payload size of the SRTP (or SRTCP) packet itself. To remain below the network path MTU, senders SHOULD constrain the SRTP (or SRTCP) payload size by this length of the Full EKT Field.

EKT can be transported over SRTCP, but some of the information that it conveys is used for SRTP processing; some elements of the EKT parameter set apply to both SRTP and SRTCP. Furthermore, SRTCP packets can be lost and both SRTP and SRTCP packets may be delivered out of order. This can lead to various race conditions if EKT is transported over SRTCP but not SRTP, which we review below.

The ROC signaled via EKT over SRTCP may be off by one when it is received by the other party(ies) in the session. In order to deal with this, receivers should simply follow the SRTP packet index estimation procedures defined in Section 3.3.1 [RFC3711].

## 2.6. Timing and Reliability Consideration

A system using EKT has the SRTP master keys distributed with EKT, rather than with call signaling. A receiver can immediately decrypt an SRTP (or SRTCP packet) using that new key, provided the SRTP packet (or SRTCP packet) also contains a Full EKT Field.

This section describes how to reliably and expediently deliver new SRTP master keys to receivers.

There are three cases to consider. The first case is a new sender joining a session which needs to communicate its SRTP master key to all the receivers. The second case is a sender changing its SRTP master key which needs to be communicated to all the receivers. The third case is a new receiver joining a session already in progress which needs to know the sender's SRTP master key.

**New sender:** A new sender SHOULD send a packet containing the Full EKT Field as soon as possible, always before or coincident with sending its initial SRTP packet. To accommodate packet loss, it is RECOMMENDED that three consecutive packets contain the Full EKT Field be transmitted. Inclusion of that Full EKT Field can be stopped early if the sender determines all receivers have received the new SRTP master key by receipt of an SRTCP receiver report or explicit ACK for a sequence number with the new key.

**Rekey:** By sending EKT over SRTP, the rekeying event shares fate with the SRTP packets protected with that new SRTP master key. To avoid sending large SRTP packets (such as video key frames) with the Full EKT Field, it can be advantageous to send smaller SRTP packets with the Full EKT Field with the Initial Sequence Number prior to the actual rekey event, but this does eliminate the benefits of fate-sharing EKT with the SRTP packets with the new SRTP master key, which increases the chance a new receiver won't have seen the new SRTP master key.

**New receiver:** When a new receiver joins a session it does not need to communicate its sending SRTP master key (because it is a receiver). When a new receiver joins a session the sender is generally unaware of the receiver joining the session. Thus, senders SHOULD periodically transmit the Full EKT Field. That interval depends on how frequently new receivers join the session, the acceptable delay before those receivers can start processing SRTP packets, and the acceptable overhead of sending the Full EKT Field. The RECOMMENDED frequency is the same as the key frame frequency if sending video or every 5 seconds. When joining a session it is likely that SRTP or SRTCP packets might be received before a packet containing the Full EKT Field is received. Thus, to avoid doubling the authentication

effort, an implementation joining an EKT session SHOULD buffer received SRTP and SRTCP packets until it receives the Full EKT Field packet and use the information in that packet to authenticate and decrypt the received SRTP/SRTCP packets.

### 3. Use of EKT with SDP Security Descriptions

The SDP Security Descriptions (SDESC) [RFC4568] specification defines a generic framework for negotiating security parameters for media streams negotiated via the Session Description Protocol with the "crypto" attribute and the Offer/Answer procedures defined in [RFC3264]. In addition to the general framework, SDESC also defines how to use that framework specifically to negotiate security parameters for Secure RTP. Below, we first provide a brief recap of the crypto attribute when used for SRTP and we then explain how it is complementary to EKT. In the rest of this Section, we provide extensions to the crypto attribute and associated offer/answer procedures to define its use with EKT.

#### 3.1. SDP Security Descriptions Recap

The SRTP crypto attribute defined for SDESC contains a tag followed by three types of parameters (refer to [RFC4568] for details):

- o Crypto-suite. Identifies the encryption and authentication transform.
- o Key parameters. SRTP keying material and parameters.
- o Session parameters. Additional (optional) SRTP parameters such as Key Derivation Rate, Forward Error Correction Order, use of unencrypted SRTP, and other parameters defined by SDESC.

The crypto attributes in the example SDP in Figure 5 illustrate these parameters.

```
v=0
o=sam 2890844526 2890842807 IN IP4 192.0.2.5
s=SRTP Discussion
i=A discussion of Secure RTP
u=http://www.example.com/seminars/srtp.pdf
e=marge@example.com (Marge Simpson)
c=IN IP4 192.0.2.12
t=2873397496 2873404696
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_80
    inline:WVNfx19zZWljdGwgKCKgewkyMjA7fQp9CnVubGVz|2^20
    FEC_ORDER=FEC_S RTP
a=crypto:2 F8_128_HMAC_SHA1_80
    inline:MTIzNDU2Nzg5QUJDREUwMTIzNDU2Nzg5QUJjiikt|2^20
    FEC_ORDER=FEC_S RTP
```

Figure 5: SDP Security Descriptions example

For legibility the SDP shows line breaks that are not present on the wire.

The first crypto attribute has the tag "1" and uses the crypto-suite AES\_CM\_128\_HMAC\_SHA1\_80. The "inline" parameter provides the SRTP master key and salt and the master key lifetime (number of packets). Finally, the FEC\_ORDER session parameter indicates the order of Forward Error Correction used (FEC is applied before SRTP processing by the sender of the SRTP media).

The second crypto attribute has the tag "2", the crypto-suite F8\_128\_HMAC\_SHA1\_80, a SRTP master key, and its associated salt. Finally, the FEC\_ORDER session parameter indicates the order of Forward Error Correction used.

### 3.2. Relationship between EKT and SDESC

SDP Security Descriptions [RFC4568] define a generic framework for negotiating security parameters for media streams negotiated via the Session Description Protocol by use of the Offer/Answer procedures defined in [RFC3264]. In addition to the general framework, SDESC also defines how to use it specifically to negotiate security parameters for Secure RTP.

EKT and SDP Security Descriptions are complementary. SDP Security Descriptions can negotiate several of the SRTP security parameters (e.g., cipher and use of Master Key Identifier) as well as SRTP master keys. SDESC, however, does not negotiate SSRs and their associated Rollover Counter (ROC). Instead, SDESC relies on a so-called "late binding", where a newly observed SSRC will have its

crypto context initialized to a ROC value of zero. Clearly, this does not work for participants joining an SRTP session that has been established for a while and hence has a non-zero ROC. It is impossible to use SDESC to join an SRTP session that is already in progress. In this case, EKT on the endpoint running SDESC can provide the additional signaling necessary to communicate the ROC (Section 6.4.1 of [RFC4568]). The use of EKT solves this problem by communicating the ROC associated with the SSRC in the media plane.

SDP Security Descriptions negotiates different SRTP master keys in the send and receive direction. The offer contains the master key used by the offerer to send media, and the answer contains the master key used by the answerer to send media. Consequently, if media is received by the offerer prior to the answer being received, the offerer does not know the master key being used. Use of SDP security preconditions can solve this problem, however it requires an additional round-trip as well as a more complicated state machine. EKT solves this problem by simply sending the master key used in the media plane thereby avoiding the need for security preconditions.

If multiple crypto-suites were offered, the offerer also will not know which of the crypto-suites offered was selected until the answer is received. EKT solves this problem by using a correlator, the Security Parameter Index (SPI), which uniquely identifies each crypto attribute in the offer.

One of the primary call signaling protocols using offer/answer is the Session Initiation Protocol (SIP) [RFC3261]. SIP uses the INVITE message to initiate a media session and typically includes an offer SDP in the INVITE. An INVITE may be "forked" to multiple recipients which potentially can lead to multiple answers being received. SDESC, however, does not properly support this scenario, mainly because SDP and RTP/RTCP does not contain sufficient information to allow for correlation of an incoming RTP/RTCP packet with a particular answer SDP. Note that extensions providing this correlation do exist (e.g., Interactive Connectivity Establishment (ICE)). SDESC addresses this point-to-multipoint problem by moving each answer to a separate RTP transport address thereby turning a point-to-multipoint scenario into multiple point-to-point scenarios. There are however significant disadvantages to doing so. As long as the crypto attribute in the answer does not contain any declarative parameters that differ from those in the offer, EKT solves this problem by use of the SPI correlator and communication of the answerer's SRTP master key in EKT.

As can be seen from the above, the combination of EKT and SDESC provides a better solution to SRTP negotiation for offer/answer than either of them alone. SDESC negotiates the various SRTP crypto

parameters (which EKT does not), whereas EKT addresses some of the shortcomings of SDESC.

### 3.3. Overview of Combined EKT and SDESC Operation

We define a new session parameter to SDESC to communicate the EKT cipher, EKT key, and Security Parameter Index to the peer. The original SDESC parameters are used as defined in [RFC4568], however the procedures associated with the SRTP master key differ slightly, since both SDESC and EKT communicate an SRTP master key. In particular, the SRTP master key communicated via SDESC is used only if there is currently no crypto context established for the SSRC in question. This will be the case when an entity has received only the offer or answer, but has yet to receive a valid EKT packet from the peer. Once a valid EKT packet is received for the SSRC, the crypto context is initialized accordingly, and the SRTP master key will then be derived from the EKT packet. Subsequent offer/answer exchanges do not change this: The most recent SRTP master key negotiated via EKT will be used, or, if none is available for the SSRC in question, the most recent SRTP master key negotiated via offer/answer will be used. This is done to avoid race conditions between the offer/answer exchange and EKT, even though this breaks some offer/answer rules. Note that with the rules described in this paragraph, once a valid EKT packet has been received for a given SSRC, rekeying for that SSRC can only be done via EKT. The associated SRTP crypto parameters however can be changed via SDESC.

### 3.4. EKT Extensions to SDP Security Descriptions

In order to use EKT and SDESC in conjunction with each other, the new SDESC session parameter "EKT" is defined. It MUST NOT appear more than once in a given crypto attribute. In the Offer/Answer model, the EKT parameter is a negotiated parameter. The "EKT" session parameter consists of three parts (the formal grammar is provided in Section 3.9):

"EKT=" <EKT\_Cipher> "|" <EKT\_Key> "|" <EKT\_SPI>

Below are details on each of these attributes.

**EKT\_Cipher:** The (optional) EKT\_Cipher field defines the EKT cipher used to encrypt the EKT key within SRTP and SRTCP packets. The default value is "AESKW\_128" in accordance with Section 2.3.1. For the AES Key Wrap cipher, the values "AESKW\_128", "AESKW\_192", and "AESKW\_256" are defined for values of L=16, 24, and 32 respectively.

**EKT\_Key:** The (mandatory) EKT\_Key field is the EKT key used to encrypt the SRTP Master Key within SRTP and SRTCP packets. The value is base64 encoded with "=" padding if padding is necessary (see Section 3.2 and 4 of [RFC4648]).

**EKT\_SPI:** The (mandatory) EKT\_SPI field is the Security Parameter Index. It is encoded as an ASCII string representing the hexadecimal value of the Security Parameter Index. The SPI identifies the \*offer\* crypto attribute (including the EKT Key and Cipher) being used for the associated SRTP session. A crypto attribute corresponds to an EKT Parameter Set and hence the SPI effectively identifies a particular EKT parameter set. Note that the scope of the SPI is the SRTP session, which may or may not be limited to the scope of the associated SIP dialog. In particular, if one of the participants in an SRTP session is an SRTP translator, the scope of the SRTP session is not limited to the scope of a single SIP dialog. However, if all of the participants in the session are endpoints or mixers, the scope of the SRTP session will correspond to a single SIP dialog.

### 3.5. Offer/Answer Considerations

In this section, we provide the offer/answer procedures associated with use of the new SDESC session parameter defined in Section 3.4. Since SDESC is defined only for unicast streams, we provide only offer/answer procedures for unicast streams here as well.

#### 3.5.1. Generating the Initial Offer - Unicast Streams

When the initial offer is generated, the offerer MUST follow the steps defined in [RFC4568] Section 7.1.1 as well as the following steps.

[[Editor's Note: following paragraph would benefit from rewording.]]

For each unicast media line using Security Descriptions and where use of EKT is desired, the offerer MUST include the EKT parameter in at least one "crypto" attribute (see [RFC4568]). The EKT parameter MUST contain the EKT\_Key and EKT\_SPI fields. The EKT\_SPI field serves to identify the EKT parameter set used for a particular SRTP or SRTCP packet. Consequently, within a single media line, a given EKT\_SPI value MUST NOT be used with multiple crypto attributes. Note that the EKT parameter set to use for the session is not yet established at this point; each offered crypto attribute contains a candidate EKT parameter set. Furthermore, if the media line refers to an existing SRTP session, then any SPI values used for EKT parameter sets in that session MUST NOT be remapped to any different EKT parameter sets. When an offer describes an SRTP session that is already in progress,

the offer SHOULD use an EKT parameter set (including EKT\_SPI and EKT\_KEY) that is already in use.

As EKT is not defined for use with MKI, a "crypto" attribute containing the EKT parameter MUST NOT contain MKI.

Important Note: The scope of the offer/answer exchange is the SIP dialog(s) established as a result of the INVITE, however the scope of EKT is the direct SRTP session, i.e., all the participants that are able to receive SRTP and SRTCP packets directly. If an SRTP session spans multiple SIP dialogs, the EKT parameter sets MUST be synchronized between all the SIP dialogs where SRTP and SRTCP packets can be exchanged. In the case where the SIP entity operates as an RTP mixer (and hence re-originates SRTP and SRTCP packets with its own SSRC), this is not an issue, unless the mixer receives traffic from the various participants on the same destination IP address and port, in which case further coordination of SPI values and crypto parameters may be needed between the SIP dialogs (note that SIP forking with multiple early media senders is an example of this). However, if it operates as a transport translator (relay) then synchronized negotiation of the EKT parameter sets on *\*all\** the involved SIP dialogs will be needed. This is non-trivial in a variety of use cases, and hence use of the combined SDES/EKT mechanism with RTP translators should be considered very carefully. It should be noted, that use of SRTP with RTP translators in general should be considered very carefully as well.

The session parameter "EKT" can either be included as an optional or mandatory parameter.

### 3.5.2. Generating the Initial Answer - Unicast Streams

When the initial answer is generated, the answerer MUST follow the steps defined in [RFC4568] Section 7.1.2 as well as the following steps.

For each unicast media line using SDESC, the answerer examines the associated crypto attribute(s) for the presence of the session parameter "EKT". If a mandatory EKT parameter is included with a "crypto" attribute, the answerer MUST support those parameters in order to accept that offered crypto attribute. If an optional EKT parameter is included instead, the answerer MAY accept the offered crypto attribute without using EKT. However, doing so will prevent the offerer from processing any packets received before the answer. If no EKT parameter are included with a crypto attribute, and that crypto attribute is accepted in the answer, EKT MUST NOT be used. If

a given a crypto attribute includes a malformed EKT parameter, that crypto attribute MUST be considered invalid.

When EKT is used with SDESC, the offerer and answerer MUST use the same SRTP master salt. Thus, the SRTP key parameter(s) in the answer crypto attribute MUST use the same master salt as the one accepted from the offer.

When the answerer accepts the offered media line and EKT is being used, the crypto attribute included in the answer MUST include the same EKT parameter values as found in the accepted crypto attribute from the offerer (however, if the default EKT cipher is being used, it may be omitted). Furthermore, the EKT parameter included MUST be mandatory (i.e., no "-" prefix).

Acceptance of a crypto attribute with an EKT parameter leads to establishment of the EKT parameter set for the corresponding SRTP session. Consequently, the answerer MUST send packets in accordance with that particular EKT parameter set only. If the answerer wants to enable the offerer to process SRTP packets received by the offerer before it receives the answer, the answerer MUST NOT include any declarative session parameters that either were not present in the offered crypto attribute, or were present but with a different value. Otherwise, the offerer's view of the EKT parameter set would differ from the answerer's until the answer is received. Similarly, unless the offerer and answerer has other means for correlating an answer with a particular SRTP session, the answer SHOULD NOT include any declarative session parameters that either were not present in the offered crypto attribute, or were present but with a different value. If this recommendation is not followed and the offerer receives multiple answers (e.g., due to SIP forking), the offerer may not be able to process incoming media stream packets correctly.

### 3.5.3. Processing of the Initial Answer - Unicast Streams

When the offerer receives the answer, it MUST perform the steps in [RFC4568] Section 7.1.3 as well as the following steps for each SRTP media stream it offered with one or more crypto lines containing EKT parameters in it.

[[Editor's Note: following paragraph would benefit from rewording.]]

If the answer crypto line contains an EKT parameter, and the corresponding crypto line from the offer contained the same EKT values, use of EKT has been negotiated successfully and MUST be used for the media stream. When determining whether the values match, an optional and mandatory parameter MUST be considered equal.

Furthermore, if the default EKT cipher is being used, it MAY be either present or absent in the offer and/or answer.

If the answer crypto line does not contain an EKT parameter, then EKT MUST NOT be used for the corresponding SRTP session. Note that if the accepted crypto attribute contained a mandatory EKT parameter in the offer, and the crypto attribute in the answer does not contain an EKT parameter, then negotiation has failed (Section 5.1.3 of [RFC4568]).

If the answer crypto line contains an EKT parameter but the corresponding offered crypto line did not, or if the values don't match or are invalid, then the offerer MUST consider the crypto line invalid (see Section 7.1.3 of [RFC4568] for further operation).

The EKT parameter set is established when the answer is received, however there are a couple of special cases to consider here. First of all, if an SRTP packet containing a Full EKT Field is received prior to the answer, then the EKT parameter set is established provisionally based on the SPI included. Once the answer (which may include declarative session parameters) is received, the EKT parameter set is fully established. The second case involves receipt of multiple answers due to SIP forking. In this case, there will be multiple EKT parameter sets; one for each SRTP session. As mentioned earlier, reliable correlation of SIP dialogs to SRTP sessions requires extensions, and hence if one or more of the answers include declarative session parameters, it may be difficult to fully establish the EKT parameter set for each SRTP session. In the absence of a specific correlation mechanism, it is RECOMMENDED, that such correlation be done based on the signaled receive IP-address in the SDP and the observed source IP-address in incoming SRTP/SRTCP packets, and, if necessary, the signaled receive UDP port and the observed source UDP port.

### 3.6. SRTP-Specific Use Outside Offer/Answer

Security Descriptions use for SRTP is not defined outside offer/answer and hence neither does Security Descriptions with EKT.

### 3.7. Modifying the Session

When a media stream using the SRTP security descriptions has been established, and a new offer/answer exchange is performed, the offerer and answerer MUST follow the steps in Section 7.1.4 of [RFC4568] as well as the following steps. SDESC allows for all parameters of the session to be modified, and the EKT session parameter are no exception to that, however, there are a few additional rules to be adhered to when using EKT.

It is permissible to start a session without the use of EKT, and then subsequently start using EKT, however the converse is not. Thus, once use of EKT has been negotiated on a particular media stream, EKT MUST continue to be used on that media stream in all subsequent offer/answer exchanges.

The reason for this is that both SDESC and EKT communicate the SRTP master key with EKT communicated master keys taking precedence. Reverting back to an SDESC-controlled master key in a synchronized manner is difficult.

Once EKT is being used, the salt for the direct SRTP session MUST NOT be changed. Thus, a new offer/answer which does not create a new SRTP session (e.g., because it reuses the same IP address and port) MUST use the same salt for all crypto attributes as is currently used for the direct SRTP session.

[[Editor's Note: following paragraph would benefit from re-arranging into earlier-described steps.]]

Finally, subsequent offer/answer exchanges MUST NOT remap a given SPI value to a different EKT parameter set until  $2^{15}$  other mappings have been used within the SRTP session. In practice, this requirement is most easily met by using a monotonically increasing SPI value (modulo  $2^{15}$  and starting with zero) per direct SRTP session. Note that a direct SRTP session may span multiple SIP dialogs, and in such cases coordination of SPI values across those SIP dialogs will be required. In the simple point-to-point unicast case without translators, the requirement simply applies within each media line in the SDP. In the point-to-multipoint case, the requirement applies across all the associated SIP dialogs.

### 3.8. Backwards Compatibility Considerations

Backwards compatibility can be achieved in a couple of ways. First of all, Security Descriptions allows for session parameters to be prefixed with "-" to indicate that they are optional. If the answerer does not support the EKT session parameter, such optional parameters will simply be ignored. When the answer is received, absence of the parameter will indicate that EKT is not being used. Receipt of SRTP or SRTCP packets prior to receipt of such an answer will obviously be problematic (as is normally the case for Security Descriptions without EKT).

Alternatively, Security Descriptions allows for multiple crypto lines to be included for a particular media stream. Thus, two crypto lines that differ in their use of EKT parameters (presence in one, absence in the other) can be used as a way to negotiate use of EKT. When the

answer is received, the accepted crypto attribute will indicate whether EKT is being used or not.

### 3.9. Grammar

The ABNF [RFC5234] syntax for the one new SDP Security Descriptions session parameter, EKT, comprising three parts is shown in Figure 6.

```

ekt      = "EKT=" cipher "|" key "|" spi
cipher   = cipher-ext / "AESKW_128" / "AESKW_192" / "AESKW_256"
cipher-ext = 1*64(ALPHA / DIGIT / "_")
key      = 1*(base64) ; See Section 4 of [RFC4648]
base64   = ALPHA / DIGIT / "+" / "/" / "="
spi      = 4HEXDIG ; See [RFC5234]
```

Figure 6: ABNF for the EKT session parameters

Using the example from Figure 6 with the EKT extensions to SDP Security Descriptions results in the following example SDP:

```

v=0
o=sam 2890844526 2890842807 IN IP4 192.0.2.5
s=SRTP Discussion
i=A discussion of Secure RTP
u=http://www.example.com/seminars/srtp.pdf
e=marge@example.com (Marge Simpson)
c=IN IP4 192.0.2.12
t=2873397496 2873404696
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_80
    inline:WVNfX19zZW1jdGwgKCKgewkyMjA7fQp9CnVubGVz|2^20
    FEC_ORDER=FEC_SRTP EKT=AESKW_128|WWVzQUxvdmVseUVLVGtleQ|AAE0
a=crypto:2 F8_128_HMAC_SHA1_80
    inline:MTIzNDU2Nzg5QUJDREUwMTIzNDU2Nzg5QUJjZGVm|2^20
    FEC_ORDER=FEC_SRTP EKT=AESKW_128|VHdvTG92ZWx5RUtUa2V5cw|AAE1
```

For legibility the SDP shows line breaks that are not present on the wire.

Figure 7: SDP Security Descriptions example with EKT

## 4. Use of EKT with DTLS-SRTP

This document defines an extension to DTLS-SRTP called Key Transport. The EKT with the DTLS-SRTP Key Transport enables secure transport of EKT keying material from one DTLS-SRTP peer to another. This enables those peers to process EKT keying material in SRTP (or SRTCP) and retrieve the embedded SRTP keying material. This combination of

protocols is valuable because it combines the advantages of DTLS (strong authentication of the endpoint and flexibility) with the advantages of EKT (allowing secure multiparty RTP with loose coordination and efficient communication of per-source keys).

#### 4.1. DTLS-SRTP Recap

DTLS-SRTP [RFC5764] uses an extended DTLS exchange between two peers to exchange keying material, algorithms, and parameters for SRTP. The SRTP flow operates over the same transport as the DTLS-SRTP exchange (i.e., the same 5-tuple). DTLS-SRTP combines the performance and encryption flexibility benefits of SRTP with the flexibility and convenience of DTLS-integrated key and association management. DTLS-SRTP can be viewed in two equivalent ways: as a new key management method for SRTP, and a new RTP-specific data format for DTLS.

#### 4.2. EKT Extensions to DTLS-SRTP

This document adds a new TLS negotiated extension called "ekt". This adds a new TLS content type, EKT, and a new negotiated extension EKT. The negotiated extension MUST only be requested in conjunction with the "use\_srtp" extension (Section 3.2 of [RFC5764]). The DTLS server MUST include "dtls-srtp-ekt" in its SDP (as a session or media level attribute) and "ekt" in its TLS ServerHello message. If a DTLS client includes "ekt" in its ClientHello, but does not receive "ekt" in the ServerHello, the DTLS client MUST NOT send DTLS packets with the "ekt" content-type.

The formal description of the dtls-srtp-ekt attribute is defined by the following ABNF [RFC5234] syntax:

```
attribute = "a=dtls-srtp-ekt"
```

Using the syntax described in DTLS [RFC6347], the following structures are used:

```
enum {
    ekt_key(0),
    ekt_key_ack(1),
    ekt_key_error(254),
    (255)
} SRTPKeyTransportType;

struct {
    SRTPKeyTransportType keytrans_type;
    uint24 length;
    uint16 message_seq;
    uint24 fragment_offset;
    uint24 fragment_length;
    select (SRTPKeyTransportType) {
        case ekt_key:
            EKTkey;
    };
} KeyTransport;

enum {
    RESERVED(0),
    AESKW_128(1),
    AESKW_192(2),
    AESKW_256(3),
} ektcipher;

struct {
    ektcipher EKT_Cipher;
    uint EKT_Key_Value<1..256>;
    uint EKT_Master_Salt<1..256>;
    uint16 EKT_SPI;
} EKTkey;
```

Figure 8: Additional TLS Data Structures

The diagram below shows a message flow of DTLS client and DTLS server using the DTLS-SRTP Key Transport extension. SRTP packets exchanged prior to the `ekt_message` are encrypted using the SRTP master key derived from the normal DTLS-SRTP key derivation function. After the `ekt_key` message, they can be encrypted using the SRTP key carried by EKT.

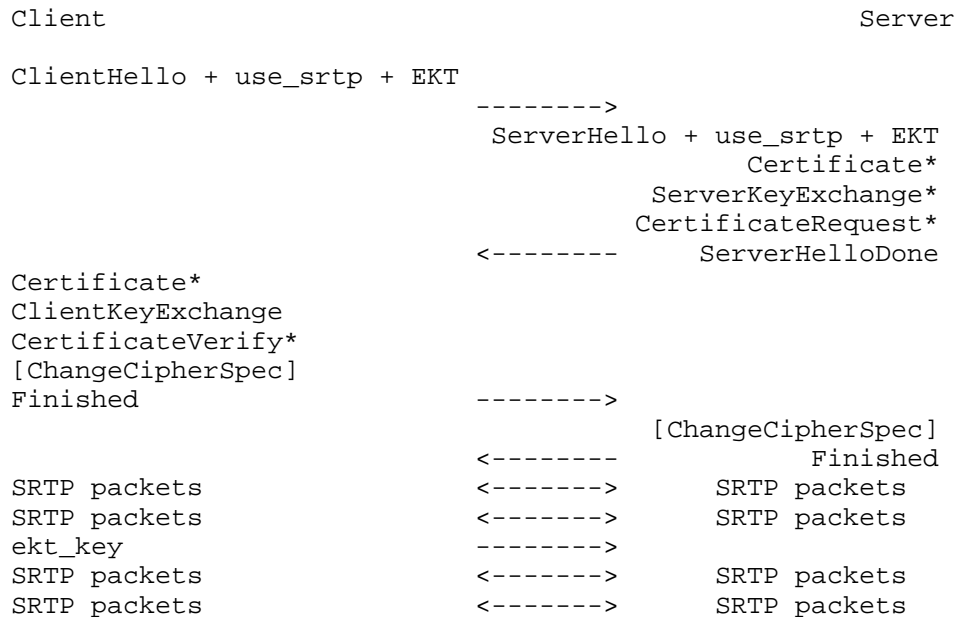


Figure 9: Handshake Message Flow

#### 4.3. Offer/Answer Considerations

This section describes Offer/Answer considerations for the use of EKT together with DTLS-SRTP for unicast and multicast streams. The offerer and answerer MUST follow the procedures specified in [RFC5764] as well as the following ones.

As most DTLS-SRTP processing is performed on the media channel, rather than in SDP, there is little processing performed in SDP other than informational and to redirect DTLS-SRTP to an alternate host. Advertising support for the extension is necessary in SDP because in some cases it is required to establish an SRTP call. For example, a mixer may be able to only support SRTP listeners if those listeners implement DTLS Key Transport (because it lacks the CPU cycles necessary to encrypt SRTP uniquely for each listener).

##### 4.3.1. Generating the Initial Offer

The initial offer contains a new SDP attribute, "dtls-srtp-ekt", which contains no value. This attribute MUST only appear at the media level. This attribute indicates the offerer is capable of supporting DTLS-SRTP with EKT extensions, and indicates the desire to use the "ekt" extension during the DTLS-SRTP handshake.

An example of SDP containing the dtls-srtp-ekt attribute::

```
v=0
o=sam 2890844526 2890842807 IN IP4 192.0.2.5
s=SRTP Discussion
i=A discussion of Secure RTP
u=http://www.example.com/seminars/srtp.pdf
e=marge@example.com (Marge Simpson)
c=IN IP4 192.0.2.12
t=2873397496 2873404696
m=audio 49170 UDP/TLS/RTP/SAVP 0
a=fingerprint:SHA-1
  4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=dtls-srtp-ekt
```

For legibility the SDP shows line breaks that are not present on the wire.

#### 4.3.2. Generating the Initial Answer

Upon receiving the initial offer, the presence of the dtls-srtp-ekt attribute indicates a desire to receive the EKT extension in the DTLS-SRTP handshake. DTLS messages should be constructed according to those two attributes.

If the answerer does not wish to perform EKT, it MUST NOT include a=dtls-srtp-ekt in the SDP answer, and it MUST NOT negotiate EKT during its DTLS-SRTP exchange.

Otherwise, the dtls-srtp-ekt attribute SHOULD be included in the answer, and EKT SHOULD be negotiated in the DTLS-SRTP handshake.

#### 4.3.3. Processing the Initial Answer

The presence of the dtls-srtp-ekt attribute indicates a desire by the answerer to perform DTLS-SRTP with EKT extensions. There are two indications the remote peer does not want to do EKT: the dtls-srtp-ekt attribute is not present in the answer, or the DTLS-SRTP exchange fails to negotiate the EKT extension. If the dtls-srtp-ekt attribute is not present in the answer, the DTLS-SRTP exchange MUST NOT attempt to negotiate the EKT extension. If the dtls-srtp-ekt attribute is present in the answer but the DTLS-SRTP exchange fails to negotiate the EKT extension, EKT MUST NOT be used with that media stream.

After successful DTLS negotiation of the EKT extension, the DTLS client and server MAY exchange SRTP packets, encrypted using the KDF described in [RFC5764]. This is normal and expected, even if Key Transport was negotiated by both sides, as neither side may (yet)

have a need to alter the SRTP key. However, it is also possible that one (or both) peers will immediately send an EKT packet before sending any SRTP, and also possible that SRTP, encrypted with an unknown key, may be received before the EKT packet is received.

#### 4.3.4. Sending DTLS EKT Key Reliably

In the absence of a round trip time estimate, the DTLS `ekt_key` message is sent using an exponential backoff initialized to 250ms, so that if the first message is sent at time 0, the next transmissions are at 250ms, 500ms, 1000ms, and so on. If a recent round trip time estimate is available, exponential backoff is used with the first transmission at 1.5 times the round trip time estimate. In either case, re-transmission stops when `ekt_key_ack` or `ekt_key_error` message is received for the matching `message_seq`.

#### 4.3.5. Modifying the Session

As DTLS-SRTP-EKT processing is done on the DTLS-SRTP channel (media channel) rather than signaling, no special processing for modifying the session is necessary.

If the initial offer and initial answer both contained EKT attributes (indicating the answerer desired to perform EKT), a subsequent offer/answer exchange MUST also contain those same EKT attributes. If not, operation is undefined and the session MAY be terminated. If the initial offer and answer failed to negotiate EKT (that is, the answer did not contain EKT attributes), EKT negotiation failed and a subsequent offer SHOULD NOT include EKT attributes.

### 5. Use of EKT with MIKEY

The advantages outlined in Section 1 are useful in some scenarios in which MIKEY is used to establish SRTP sessions. In this section, we briefly review MIKEY and related work, and discuss these scenarios.

An SRTP sender or a group controller can use MIKEY to establish a SRTP cryptographic context. This capability includes the distribution of a TEK generation key (TGK) or the TEK itself, security policy payload, crypto session bundle ID (CSB\_ID) and a crypto session ID (CS\_ID). The TEK directly maps to an SRTP master key, whereas the TGK is used along with the CSB\_ID and a CS\_ID to generate a TEK. The CS\_ID is used to generate multiple TEKs (SRTP master keys) from a single TGK. For a media stream in SDP, MIKEY allocates two consecutive numbers for the crypto session IDs, so that each direction uses a different SRTP master key (see [RFC4567]).

The MIKEY specification [RFC3830] defines three modes to exchange keys, associated parameters and to protect the MIKEY message: pre-shared key, public-key encryption and Diffie-Hellman key exchange. In the first two modes the MIKEY initiator only chooses and distributes the TKG or TEK, whereas in the third mode both MIKEY entities (the initiator and responder) contribute to the keys. All three MIKEY modes have in common that for establishing a SRTP session the exchanged key is valid for the send and receive direction. Especially for group communications it is desirable to update the SRTP master key individually per direction. EKT provides this property by distributing the SRTP master key within the SRTP/SRTCP packet.

MIKEY already supports synchronization of ROC values between the MIKEY initiator and responder. The SSRC / ROC value pair is part of the MIKEY Common Header payload. This allows providing the current ROC value to late joiners of a session. However, in some scenarios a key management based ROC synchronization is not sufficient. For example, in mobile and wireless environments, members may go in and out of coverage and may miss a sequence number overrun. In point-to-multipoint translator scenarios it is desirable to not require the group controller to track the ROC values of each member, but to provide the ROC value by the originator of the SRTP packet. A better alternative to synchronize the ROC values is to send them directly via SRTP/SRTCP as EKT does. A separate SRTP extension [RFC4771] includes the ROC in a modified authentication tag but that extension does not support updating the SRTP master key.

Besides the ROC, MIKEY synchronizes also the SSRC values of the SRTP streams. Each sender of a stream sends the associated SSRC within the MIKEY message to the other party. If an SRTP session participant starts a new SRTP source (SSRC) or a new participant is added to a group, subsequent SDP offer/answer and MIKEY exchanges are necessary to update the SSRC values. EKT improves these scenarios by updating the keys and SSRC values without coordination on the signaling channel. With EKT, SRTP can handle early media, since the EKT SPI allows the receiver to identify the cryptographic keys and parameters used by the source.

The MIKEY specification [RFC3830] suggests the use of unicast for rekeying. This method does not scale well to large groups or interactive groups. The EKT extension of SRTP/SRTCP provides a solution for rekeying the SRTP master key and for ROC/SSRC synchronization. EKT is not a substitution for MIKEY, but rather a complementary addition to address the above described limitations of MIKEY.

In the next section we provide an extension to MIKEY for support of EKT. EKT can be used only with the pre-shared key or public-key encryption MIKEY mode of [RFC3830]. The Diffie-Hellman exchange mode is not suitable in conjunction with EKT, because it is not possible to establish one common EKT key over multiple EKT entities. Additional MIKEY modes specified in separate documents are not considered for EKT.

### 5.1. EKT Extensions to MIKEY

In order to use EKT with MIKEY, the EKT cipher, EKT key and EKT SPI is negotiated in the MIKEY message exchange.

The following parameters are added to the MIKEY Security Protocol Parameters namespace ([RFC3830], Section 6.10.1). (TBD will be requested from IANA [NOTE TO RFC EDITOR])

Type	Meaning	Possible values
TBD	EKT cipher	see below
TBD	EKT SPI	a 15-bit value

Figure 10: MIKEY Security Protocol Parameters

EKT cipher	Value
(reserved)	0
AESKW_128	1
AESKW_192	2
AESKW_256	3

Figure 11: EKT Cipher Parameters

EKT\_Key is transported in the MIKEY KEMAC payload within one separate Key Data sub-payload. As specified in Section 6.2 of [RFC3830], the KEMAC payload carries the TEK Generation Key (TGK) or the Traffic Encryption Key (TEK). One or more TGKs or TEKs are carried in individual Key Data sub-payloads within the KEMAC payload. The KEMAC payload is encrypted as part of MIKEY. The Key Data sub-payload, specified in Section 6.13 of [RFC3830], has the following format:

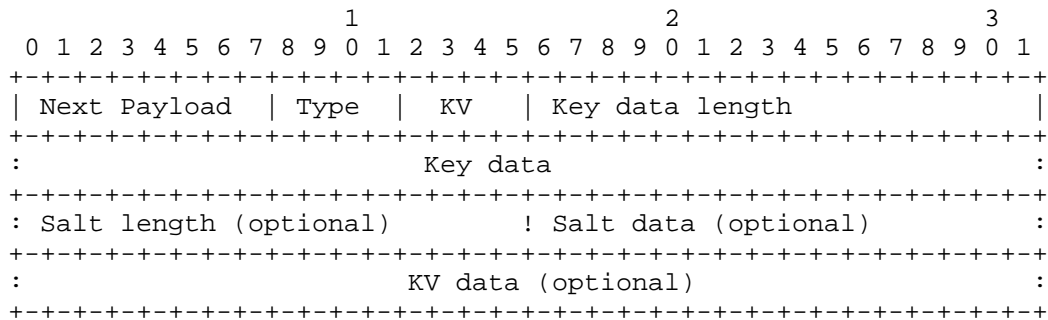


Figure 12: Key Data Sub-Payload of MIKEY

These fields are described below:

**Type:** 4 bits in length, indicates the type of key included in the payload. We define Type = TBD (will be requested from IANA [NOTE TO RFC EDITOR]) to indicate transport of the EKT key.

**KV:** (4 bits): indicates the type of key validity period specified. KV=1 is currently specified as an SPI. We use that value to indicate the KV data contains the EKT\_SPI for the key type EKT\_Key. KV data would be 16 bits in length, but it is also possible to interpret the length from the 'Key data len' field. KV data MUST be present for the key type EKT\_Key when KV=1.

**Salt length, Salt Data:** These optional fields SHOULD be omitted for the key type EKT\_Key, if the SRTP master salt is already present in the TKG or TEK Key Data sub-payload. The EKT\_Key sub-payload MUST contain a SRTP master salt, if the SRTP master salt is not already present in the TKG or TEK Key Data sub-payload.

**KV Data:** length determined by Key Data Length field.

## 5.2. Offer/Answer Considerations

This section describes Offer/Answer considerations for the use of EKT together with MIKEY for unicast streams. The offerer and answerer MUST follow the procedures specified in [RFC3830] and [RFC4567] as well as the following ones.

### 5.2.1. Generating the Initial Offer

If it is intended to use MIKEY together with EKT, the offerer MUST include at least one MIKEY key-mgmt attribute with one EKT\_Key Key Data sub-payload and the SRTP Security Policy payload (SP) with the policy parameter EKT SPI. The policy parameter EKT Cipher is

OPTIONAL, The default value is "AESKW\_128" in accordance with Section 2.3.1. MIKEY can be used on session or media level. On session level, MIKEY provides the keys for multiple SRTP sessions in the SDP offer. The EKT SPI references a EKT parameter set including the Secure RTP parameters as specified in Section 8.2 in [RFC3711]. If MIKEY is used on session level, it is only possible to use one EKT SPI value. Therefore, the session-level MIKEY message MUST contain one SRTP Security Policy payload only, which is valid for all related SRTP media lines. If MIKEY is used on media level, different SRTP Security Policy parameters (and consequently different EKT SPI values) can be used for each media line. If MIKEY is used on session and media level, the media level content overrides the session level content.

EKT requires a single shared SRTP master salt between all participants in the direct SRTP session. If a MIKEY key-mgmt attribute contains more than one TKG or TEK Key Data sub-payload, all the sub-payloads MUST contain the same master salt value. Consequently, the EKT\_Key Key Data sub-payload MAY also contain the same salt or MAY omit the salt value. If the SRTP master salt is not present in the TKG and TEK Key Data sub-payloads, the EKT\_Key sub-payload MUST contain a master salt.

#### 5.2.2. Generating the Initial Answer

For each media line in the offer using MIKEY, provided on session and/or on media level, the answerer examines the related MIKEY key-mgmt attributes for the presence of EKT parameters. In order to accept the offered key-mgmt attribute, the MIKEY message MUST contain one EKT\_Key Key Data sub-payload and the SRTP Security Policy payload with policy parameter EKT SPI. The answerer examines also the existence of a SRTP master salt in the TKG/TEK and/or the EKT\_Key sub-payloads. If multiple salts are available, all values MUST be equal. If the salt values differ or no salt is present, the key-mgmt attribute MUST be considered as invalid.

The MIKEY responder message in the SDP answer does not contain a MIKEY KEMAC or Security Policy payload and consequently does not contain any EKT parameters. If a key-mgmt attribute for a media line was accepted by the answerer, the EKT parameter set of the offerer is valid for both directions of the SRTP session.

#### 5.2.3. Processing the Initial Answer

On reception of the answer, the offerer examines if EKT has been accepted for the offered media lines. If a MIKEY key-mgmt attribute is received containing a valid MIKEY responder message, EKT has been successfully negotiated. On receipt of a MIKEY error message, EKT

negotiation has failed. For example, this may happen if an EKT extended MIKEY initiator message is sent to a MIKEY entity not supporting EKT. A MIKEY error code 'Invalid SPpar' or 'Invalid DT' is returned to indicate that the EKT parameters (EKT Cipher and EKT SPI) in the SRTP Security Policy payload or the EKT\_Key sub-payload is not supported. In this case, the offerer may send a second SDP offer with a MIKEY key-mgmt attribute without the additional EKT extensions.

This behavior can be improved by offering two key-mgmt SDP attributes. One attribute offers MIKEY with SRTP and EKT and the other attribute offers MIKEY with SRTP without EKT.

#### 5.2.4. Modifying the Session

Once an SRTP stream has been established, a new offer/answer exchange can modify the session including the EKT parameters. If the EKT key or EKT cipher is modified (i.e., a new EKT parameter set is created) the offerer MUST also provide a new EKT SPI value. The offerer MUST NOT remap an existing EKT SPI value to a new EKT parameter set. Similar, a modification of the SRTP Security Policy leads to a new EKT parameter set and requires a fresh EKT SPI, even if the EKT key or cipher did not change.

Once EKT is being used, the SRTP master salt for the SRTP session MUST NOT be changed. The salt in the Key Data sub-payloads within the subsequent offers MUST be the same as the one already used.

After EKT has been successfully negotiated for a session and an SRTP master key has been transported by EKT, it is difficult to switch back to a pure MIKEY based key exchange in a synchronized way. Therefore, once EKT is being used for a session, EKT MUST be used also in all subsequent offer/answer exchanges for that session.

### 6. Using EKT for Interoperability between Key Management Systems

A media gateway (MGW) can provide interoperability between an SRTP-EKT endpoint and a non-EKT SRTP endpoint. When doing this function, the MGW can perform non-cryptographic transformations on SRTP packets outlined above. However, there are some uses of cryptography that will be required for that gateway. If a new SRTP master key is communicated to the MGW (via EKT from the EKT leg, or via Security Descriptions without EKT from the Security Descriptions leg), the MGW needs to convert that information for the other leg, and that process will incur some cryptographic operations. Specifically, if the new key arrived via EKT, the key must be decrypted and then sent in Security Descriptions (e.g., as a SIP re-INVITE); likewise, if a new

key arrives via Security Descriptions that must be encrypted via EKT and sent in SRTP/SRTCP.

Additional non-normative information can be found in Appendix A.

## 7. Design Rationale

From [RFC3550], a primary function of RTCP is to carry the CNAME, a "persistent transport-level identifier for an RTP source" since "receivers require the CNAME to keep track of each participant." EKT works in much the same way but uses SRTP to carry information needed for the proper processing of the SRTP traffic.

With EKT, SRTP gains the ability to synchronize the creation of cryptographic contexts across all of the participants in a single session. This feature provides some, but not all, of the functionality that is present in IKE phase two (but not phase one). Importantly, EKT does not provide a way to indicate SRTP options.

With EKT, external signaling mechanisms provide the SRTP options and the EKT Key, but need not provide the key(s) for each individual SRTP source. EKT provides a separation between the signaling mechanisms and the details of SRTP. The signaling system need not coordinate all SRTP streams, nor predict in advance how many sources will be present, nor communicate SRTP-level information (e.g., rollover counters) of current sessions.

EKT is especially useful for multi-party sessions, and for the case where multiple RTP sessions are sent to the same destination transport address (see the example in the definition of "RTP session" in [RFC3550]). A SIP offer that is forked in parallel (sent to multiple endpoints at the same time) can cause multiple RTP sessions to be sent to the same transport address, making EKT useful for use with SIP.

EKT can also be used in conjunction with a scalable group-key management system like GDOI [RFC6407]. In such a combination GDOI would provide a secure entity authentication method for group members, and a scalable way to revoke group membership; by itself, EKT does not attempt to provide either capability.

EKT carries the encrypted key in a new SRTP field (at the end of the SRTP packet). This maintains compatibility with the existing SRTP specification by defining a new crypto function that incorporates the encrypted key, and a new authentication transform to provide implicit authentication of the encrypted key.

The main motivation for the use of the variable-length EKT format is bandwidth conservation. When EKT is sent over SRTP, there will be a loss of (usable) bandwidth due to the additional EKT bytes in each RTP packet. For some applications, this bandwidth loss is significant.

### 7.1. Alternatives

In its current design, EKT requires that the Master Salt be established out of band. That requirement is undesirable. In an offer/answer environment, it forces the answerer to re-use the same Master Salt value used by the offerer. The Master Salt value could be carried in EKT packets though that would consume yet more bandwidth.

In some scenarios, two SRTP sessions may be combined into a single session. When using EKT in such sessions, it is desirable to have an SPI value that is larger than 15 bits, so that collisions between SPI values in use in the two different sessions are unlikely (since each collision would confuse the members of one of the sessions).

An alternative that addresses both of these needs is as follows: the SPI value can be lengthed from 15 bits to 63 bits, and the Master Salt can be identical to, or constructed from, the SPI value. SRTP conventionally uses a 14-byte Master Salt, but shorter values are acceptable. This alternative would add six bytes to each EKT packet; that overhead may be a reasonable tradeoff for addressing the problems outlined above. This is considered too high a bandwidth penalty.

## 8. Security Considerations

EKT inherits the security properties of the SRTP keying it uses: Security Descriptions, DTLS-SRTP, or MIKEY.

With EKT, each SRTP sender and receiver MUST generate distinct SRTP master keys. This property avoids any security concern over the re-use of keys, by empowering the SRTP layer to create keys on demand. Note that the inputs of EKT are the same as for SRTP with key-sharing: a single key is provided to protect an entire SRTP session. However, EKT remains secure even in the absence of out-of-band coordination of SSRCs, and even when SSRC values collide.

The EKT Cipher includes its own authentication/integrity check. For an attacker to successfully forge a full EKT packet, it would need to defeat the authentication mechanisms of both the EKT Cipher and the SRTP authentication mechanism.

The presence of the SSRC in the EKT\_Plaintext ensures that an attacker cannot substitute an EKT\_Ciphertext from one SRTP stream into another SRTP stream.

An attacker who strips a Full\_EKT\_Field from an SRTP packet may prevent the intended receiver of that packet from being able to decrypt it. This is a minor denial of service vulnerability. Similarly, an attacker who adds a Full\_EKT\_Field can disrupt service.

An attacker could send packets containing either Short EKT Field or Full EKT Field, in an attempt to consume additional CPU resources of the receiving system. In the case of the Short EKT Field, this field is stripped and normal SRTP or SRTCP processing is performed. In the case of the Full EKT Field, the attacker would have to have guessed or otherwise determined the SPI being used by the receiving system. If an invalid SPI is provided by the attacker, processing stops. If a valid SPI is provided by the attacker, the receiving system will decrypt the EKT ciphertext and return an authentication failure (Step 3 of Section 2.2.2).

EKT can rekey an SRTP stream until the SRTP rollover counter (ROC) needs to roll over. EKT does not extend SRTP's rollover counter (ROC), and like SRTP itself EKT cannot properly handle a ROC rollover. Thus even if using EKT, new (master or session) keys need to be established after  $2^{48}$  packets are transmitted in a single SRTP stream as described in Section 3.3.1 of [RFC3711]. Due to the relatively low packet rates of typical RTP sessions, this is not expected to be a burden.

The confidentiality, integrity, and authentication of the EKT cipher MUST be at least as strong as the SRTP cipher.

Part of the EKT\_Plaintext is known, or easily guessable to an attacker. Thus, the EKT Cipher MUST resist known plaintext attacks. In practice, this requirement does not impose any restrictions on our choices, since the ciphers in use provide high security even when much plaintext is known.

An EKT cipher MUST resist attacks in which both ciphertexts and plaintexts can be adaptively chosen. An EKT cipher MUST resist attacks in which both ciphertexts and plaintexts can be adaptively chosen and adversaries that can query both the encryption and decryption functions adaptively.

## 9. IANA Considerations

IANA is requested to register EKT from Section 3.9 into the Session Description Protocol (SDP) Security Descriptions [iana-sdp-sdesc] registry for "SRTP Session Parameters".

IANA is requested to register the following new attributes into the SDP Attributes registry [iana-sdp-attr].

Attribute name: dtls-srtp-ekt

Long form name: DTLS-SRTP with EKT

Type of attribute: Media-level

Subject to charset: No

Purpose: Indicates support for DTLS-SRTP with EKT

Appropriate values: No values

Contact name: Dan Wing, dwing@cisco.com

We request the following IANA assignments from the existing [iana-mikey] name spaces in the IETF consensus range (0-240) [RFC3830]:

- o From the Key Data payload name spaces, a value to indicate the type as the 'EKT\_Key'.

Furthermore, we need the following two new IANA registries created, populated with the initial values in this document. New values for both of these registries can be defined via Specification Required [RFC5226].

- o EKT parameter type, initially populated with the list from Figure 10
- o EKT cipher, initially populated with the list from Figure 11

## 10. Acknowledgements

Thanks to Lakshminath Dondeti for assistance with earlier versions of this document. Thanks to Kai Fischer for writing the MIKEY section.

Thanks to Nermeen Ismail, Eddy Lem, Rob Raymond, and Yi Cheng for fruitful discussions and comments. Thanks to Felix Wyss for his review and comments regarding ciphers. Thanks to Michael Peck for

his review. Thanks to Magnus Westerlund for his review. Thanks to Michael Peck and Jonathan Lennox for their review comments.

## 11. References

### 11.1. Normative References

- [FIPS197] National Institute of Standards and Technology (NIST), "The Advanced Encryption Standard (AES)", FIPS-197 Federal Information Processing Standard, November 2001.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC4563] Carrara, E., Lehtovirta, V., and K. Norrman, "The Key ID Information Type for the General Extension Payload in Multimedia Internet KEYing (MIKEY)", RFC 4563, June 2006.
- [RFC4567] Arkko, J., Lindholm, F., Naslund, M., Norrman, K., and E. Carrara, "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)", RFC 4567, July 2006.
- [RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

## 11.2. 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, June 2002.
- [RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC4771] Lehtovirta, V., Naslund, M., and K. Norrman, "Integrity Transform Carrying Roll-Over Counter for the Secure Real-time Transport Protocol (SRTP)", RFC 4771, January 2007.
- [RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, September 2009.
- [RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation", RFC 6407, October 2011.
- [iana-mikey] IANA, , "Multimedia Internet KEYing (Mikey) Payload Name Spaces", 2011, <<http://www.iana.org/assignments/mikey-payloads/mikey-payloads.xhtml>>.
- [iana-sdp-attr] IANA, , "SDP Parameters", 2011, <<http://www.iana.org/assignments/sdp-parameters/sdp-parameters.xml>>.

```
[iana-sdp-sdesc]
    IANA, , "Session Description Protocol (SDP) Security
    Descriptions: SRTP Session Parameters", 2011,
    <http://www.iana.org/assignments/sdp-security-
    descriptions/sdp-security-descriptions.xml#sdp-security-
    descriptions-4>.
```

#### Appendix A. Using EKT to Optimize Interworking DTLS-SRTP with Security Descriptions

Today, SDP Security Descriptions [RFC4568] is used for distributing SRTP keys in several different IP PBX systems. The IP PBX systems are typically used within a single enterprise. A Session Border Controller is a reasonable solution to interwork between Security Descriptions in one network and DTLS-SRTP in another network. For example, a mobile operator (or an Enterprise) could operate Security Descriptions within their network and DTLS-SRTP towards the Internet.

However, due to the way Security Descriptions and DTLS-SRTP manage their SRTP keys, such an SBC has to authenticate, decrypt, re-encrypt, and re-authenticate the SRTP (and SRTCP) packets in one direction, as shown in Figure 13, below. This is computationally expensive.

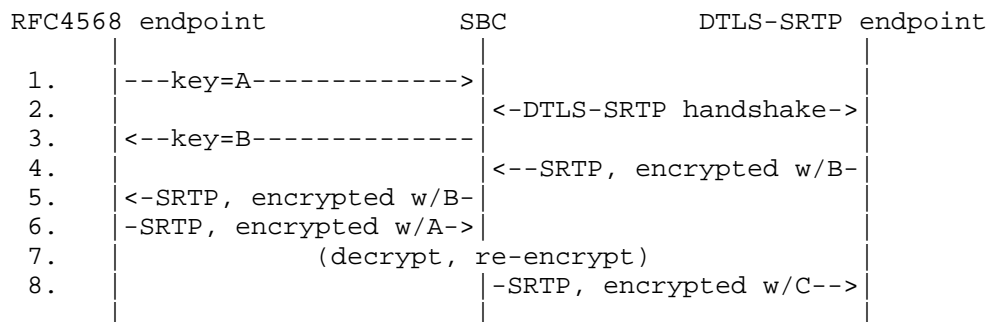


Figure 13: Interworking Security Descriptions and DTLS-SRTP

The message flow is as follows (similar steps occur with SRTCP):

1. The Security Descriptions [RFC4568] endpoint discloses its SRTP key to the SBC, using a=crypto in its SDP.
2. SBC completes DTLS-SRTP handshake. From this handshake, the SBC derives the SRTP key for traffic from the DTLS-SRTP endpoint (key B) and to the DTLS-SRTP endpoint (key C).

3. The SBC communicates the SRTP encryption key (key B) to the Security Descriptions endpoint (using a=crypto). (There is no way, with DTLS-SRTP, to communicate the Security Descriptions key to the DTLS-SRTP key endpoint.)
4. The DTLS-SRTP endpoint sends an SRTP key, encrypted with its key B. This is received by the SBC.
5. The received SRTP packet is simply forwarded; the SBC does not need to do anything with this packet as its key (key B) was already communicated in step 3.
6. The Security Descriptions endpoint sends an SRTP packet, encrypted with its key A.
7. The SBC has to authenticate and decrypt the SRTP packet (using key A), and re-encrypt it and generate an HMAC (using key C).
8. The SBC sends the new SRTP packet.

If EKT is deployed on the DTLS-SRTP endpoints, EKT helps to avoid the computationally expensive operation so the SBC does not need to perform any per-packet operations on the SRTP (or SRTCP) packets in either direction. With EKT the SBC can simply forward the SRTP (and SRTCP) packets in both directions without per-packet HMAC or cryptographic operations.

To accomplish this interworking, DTLS-SRTP EKT must be supported on the DTLS-SRTP endpoint, which allows the SBC to transport the Security Description key to the EKT endpoint and send the DTLS-SRTP key to the Security Descriptions endpoint. This works equally well for both incoming and outgoing calls. An abbreviated message flow is shown in Figure 14, below.

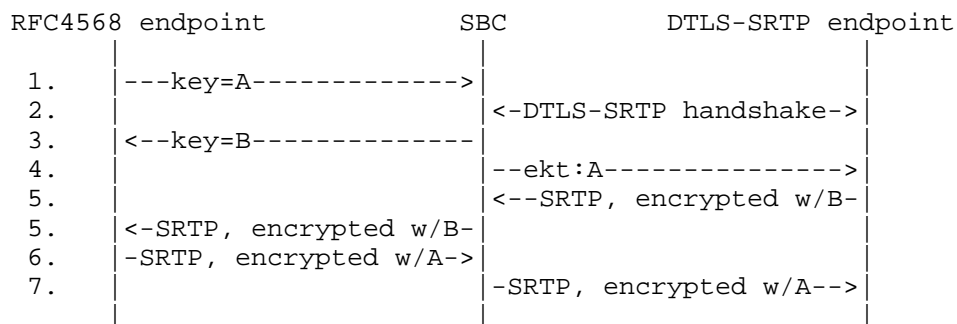


Figure 14: Interworking Security Descriptions and EKT

The message flow is as follows (similar steps occur with SRTCP):

1. Security Descriptions endpoint discloses its SRTP key to the SBC (a=crypto).
2. SBC completes DTLS-SRTP handshake. From this handshake, the SBC derives the SRTP key for traffic from the DTLS-SRTP endpoint (key B) and to the DTLS-SRTP endpoint (key C).
3. The SBC communicates the SRTP encryption key (key B) to the Security Descriptions endpoint.
4. The SBC sends an EKT packet indicating that SRTP will be encrypted with 'key A' towards the DTLS-SRTP endpoint.
5. The DTLS-SRTP endpoint sends an SRTP key, encrypted with its key B. This is received by the SBC.
6. The received SRTP packet is simply forwarded; the SBC does not need to do anything with this packet as its key (key B) was communicated in step 3.
7. The Security Descriptions endpoint sends an SRTP packet, encrypted with its key A.
8. The received SRTP packet is simply forwarded; the SBC does not need to do anything with this packet as its key (key A) was communicated in step 4.

#### Authors' Addresses

John Mattsson (editor)  
Ericsson AB  
SE-164 80 Stockholm  
Sweden

Phone: +46 10 71 43 501  
Email: john.mattsson@ericsson.com

David A. McGrew  
Cisco Systems, Inc.  
510 McCarthy Blvd.  
Milpitas, CA 95035  
US

Phone: (408) 525 8651  
Email: mcgrew@cisco.com  
URI: <http://www.mindspring.com/~dmcgrew/dam.htm>

Dan Wing  
Cisco Systems, Inc.  
510 McCarthy Blvd.  
Milpitas, CA 95035  
US

Phone: (408) 853 4197  
Email: dwing@cisco.com

Flemming Andreason  
Cisco Systems, Inc.  
499 Thornall Street  
Edison, NJ 08837  
US

Email: fandreas@cisco.com

RTCWEB Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 18, 2016

C. Perkins  
University of Glasgow  
M. Westerlund  
Ericsson  
J. Ott  
Aalto University  
March 17, 2016

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP  
draft-ietf-rtcweb-rtp-usage-26

Abstract

The Web Real-Time Communication (WebRTC) framework provides support for direct interactive rich communication using audio, video, text, collaboration, games, etc. between two peers' web-browsers. This memo describes the media transport aspects of the WebRTC framework. It specifies how the Real-time Transport Protocol (RTP) is used in the WebRTC context, and gives requirements for which RTP features, profiles, and extensions need to be supported.

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 <http://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 September 18, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Rationale . . . . .	4
3. Terminology . . . . .	4
4. WebRTC Use of RTP: Core Protocols . . . . .	5
4.1. RTP and RTCP . . . . .	5
4.2. Choice of the RTP Profile . . . . .	7
4.3. Choice of RTP Payload Formats . . . . .	8
4.4. Use of RTP Sessions . . . . .	10
4.5. RTP and RTCP Multiplexing . . . . .	10
4.6. Reduced Size RTCP . . . . .	11
4.7. Symmetric RTP/RTCP . . . . .	11
4.8. Choice of RTP Synchronisation Source (SSRC) . . . . .	12
4.9. Generation of the RTCP Canonical Name (CNAME) . . . . .	12
4.10. Handling of Leap Seconds . . . . .	13
5. WebRTC Use of RTP: Extensions . . . . .	13
5.1. Conferencing Extensions and Topologies . . . . .	14
5.1.1. Full Intra Request (FIR) . . . . .	15
5.1.2. Picture Loss Indication (PLI) . . . . .	15
5.1.3. Slice Loss Indication (SLI) . . . . .	16
5.1.4. Reference Picture Selection Indication (RPSI) . . . . .	16
5.1.5. Temporal-Spatial Trade-off Request (TSTR) . . . . .	16
5.1.6. Temporary Maximum Media Stream Bit Rate Request (TMMBR) . . . . .	16
5.2. Header Extensions . . . . .	17
5.2.1. Rapid Synchronisation . . . . .	17
5.2.2. Client-to-Mixer Audio Level . . . . .	17
5.2.3. Mixer-to-Client Audio Level . . . . .	18
5.2.4. Media Stream Identification . . . . .	18
5.2.5. Coordination of Video Orientation . . . . .	18
6. WebRTC Use of RTP: Improving Transport Robustness . . . . .	19
6.1. Negative Acknowledgements and RTP Retransmission . . . . .	19
6.2. Forward Error Correction (FEC) . . . . .	20
7. WebRTC Use of RTP: Rate Control and Media Adaptation . . . . .	20
7.1. Boundary Conditions and Circuit Breakers . . . . .	21
7.2. Congestion Control Interoperability and Legacy Systems . . . . .	22
8. WebRTC Use of RTP: Performance Monitoring . . . . .	22
9. WebRTC Use of RTP: Future Extensions . . . . .	23
10. Signalling Considerations . . . . .	23
11. WebRTC API Considerations . . . . .	25
12. RTP Implementation Considerations . . . . .	27

12.1.	Configuration and Use of RTP Sessions . . . . .	27
12.1.1.	Use of Multiple Media Sources Within an RTP Session	27
12.1.2.	Use of Multiple RTP Sessions . . . . .	28
12.1.3.	Differentiated Treatment of RTP Streams . . . . .	33
12.2.	Media Source, RTP Streams, and Participant Identification . . . . .	35
12.2.1.	Media Source Identification . . . . .	35
12.2.2.	SSRC Collision Detection . . . . .	36
12.2.3.	Media Synchronisation Context . . . . .	37
13.	Security Considerations . . . . .	37
14.	IANA Considerations . . . . .	39
15.	Acknowledgements . . . . .	39
16.	References . . . . .	39
16.1.	Normative References . . . . .	39
16.2.	Informative References . . . . .	44
	Authors' Addresses . . . . .	46

## 1. Introduction

The Real-time Transport Protocol (RTP) [RFC3550] provides a framework for delivery of audio and video teleconferencing data and other real-time media applications. Previous work has defined the RTP protocol, along with numerous profiles, payload formats, and other extensions. When combined with appropriate signalling, these form the basis for many teleconferencing systems.

The Web Real-Time communication (WebRTC) framework provides the protocol building blocks to support direct, interactive, real-time communication using audio, video, collaboration, games, etc., between two peers' web-browsers. This memo describes how the RTP framework is to be used in the WebRTC context. It proposes a baseline set of RTP features that are to be implemented by all WebRTC Endpoints, along with suggested extensions for enhanced functionality.

This memo specifies a protocol intended for use within the WebRTC framework, but is not restricted to that context. An overview of the WebRTC framework is given in [I-D.ietf-rtcweb-overview].

The structure of this memo is as follows. Section 2 outlines our rationale in preparing this memo and choosing these RTP features. Section 3 defines terminology. Requirements for core RTP protocols are described in Section 4 and suggested RTP extensions are described in Section 5. Section 6 outlines mechanisms that can increase robustness to network problems, while Section 7 describes congestion control and rate adaptation mechanisms. The discussion of mandated RTP mechanisms concludes in Section 8 with a review of performance monitoring and network management tools. Section 9 gives some guidelines for future incorporation of other RTP and RTP Control

Protocol (RTCP) extensions into this framework. Section 10 describes requirements placed on the signalling channel. Section 11 discusses the relationship between features of the RTP framework and the WebRTC application programming interface (API), and Section 12 discusses RTP implementation considerations. The memo concludes with security considerations (Section 13) and IANA considerations (Section 14).

## 2. Rationale

The RTP framework comprises the RTP data transfer protocol, the RTP control protocol, and numerous RTP payload formats, profiles, and extensions. This range of add-ons has allowed RTP to meet various needs that were not envisaged by the original protocol designers, and to support many new media encodings, but raises the question of what extensions are to be supported by new implementations. The development of the WebRTC framework provides an opportunity to review the available RTP features and extensions, and to define a common baseline RTP feature set for all WebRTC Endpoints. This builds on the past 20 years of RTP development to mandate the use of extensions that have shown widespread utility, while still remaining compatible with the wide installed base of RTP implementations where possible.

RTP and RTCP extensions that are not discussed in this document can be implemented by WebRTC Endpoints if they are beneficial for new use cases. However, they are not necessary to address the WebRTC use cases and requirements identified in [RFC7478].

While the baseline set of RTP features and extensions defined in this memo is targeted at the requirements of the WebRTC framework, it is expected to be broadly useful for other conferencing-related uses of RTP. In particular, it is likely that this set of RTP features and extensions will be appropriate for other desktop or mobile video conferencing systems, or for room-based high-quality telepresence applications.

## 3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The RFC 2119 interpretation of these key words applies only when written in ALL CAPS. Lower- or mixed-case uses of these key words are not to be interpreted as carrying special significance in this memo.

We define the following additional terms:

**WebRTC MediaStream:** The MediaStream concept defined by the W3C in the WebRTC API [W3C.WD-mediapture-streams-20130903]. A MediaStream consists of zero or more MediaStreamTracks.

**MediaStreamTrack:** Part of the MediaStream concept defined by the W3C in the WebRTC API [W3C.WD-mediapture-streams-20130903]. A MediaStreamTrack is an individual stream of media from any type of media source like a microphone or a camera, but also conceptual sources, like a audio mix or a video composition, are possible.

**Transport-layer Flow:** A uni-directional flow of transport packets that are identified by having a particular 5-tuple of source IP address, source port, destination IP address, destination port, and transport protocol used.

**Bi-directional Transport-layer Flow:** A bi-directional transport-layer flow is a transport-layer flow that is symmetric. That is, the transport-layer flow in the reverse direction has a 5-tuple where the source and destination address and ports are swapped compared to the forward path transport-layer flow, and the transport protocol is the same.

This document uses the terminology from [I-D.ietf-avtext-rtp-grouping-taxonomy] and [I-D.ietf-rtcweb-overview]. Other terms are used according to their definitions from the RTP Specification [RFC3550]. Especially note the following frequently used terms: RTP Stream, RTP Session, and Endpoint.

#### 4. WebRTC Use of RTP: Core Protocols

The following sections describe the core features of RTP and RTCP that need to be implemented, along with the mandated RTP profiles. Also described are the core extensions providing essential features that all WebRTC Endpoints need to implement to function effectively on today's networks.

##### 4.1. RTP and RTCP

The Real-time Transport Protocol (RTP) [RFC3550] is REQUIRED to be implemented as the media transport protocol for WebRTC. RTP itself comprises two parts: the RTP data transfer protocol, and the RTP control protocol (RTCP). RTCP is a fundamental and integral part of RTP, and MUST be implemented and used in all WebRTC Endpoints.

The following RTP and RTCP features are sometimes omitted in limited functionality implementations of RTP, but are REQUIRED in all WebRTC Endpoints:

- o Support for use of multiple simultaneous SSRC values in a single RTP session, including support for RTP endpoints that send many SSRC values simultaneously, following [RFC3550] and [I-D.ietf-avtcore-rtp-multi-stream]. The RTCP optimisations for multi-SSRC sessions defined in [I-D.ietf-avtcore-rtp-multi-stream-optimisation] MAY be supported; if supported the usage MUST be signalled.
- o Random choice of SSRC on joining a session; collision detection and resolution for SSRC values (see also Section 4.8).
- o Support for reception of RTP data packets containing CSRC lists, as generated by RTP mixers, and RTCP packets relating to CSRCs.
- o Sending correct synchronisation information in the RTCP Sender Reports, to allow receivers to implement lip-synchronisation; see Section 5.2.1 regarding support for the rapid RTP synchronisation extensions.
- o Support for multiple synchronisation contexts. Participants that send multiple simultaneous RTP packet streams SHOULD do so as part of a single synchronisation context, using a single RTCP CNAME for all streams and allowing receivers to play the streams out in a synchronised manner. For compatibility with potential future versions of this specification, or for interoperability with non-WebRTC devices through a gateway, receivers MUST support multiple synchronisation contexts, indicated by the use of multiple RTCP CNAMEs in an RTP session. This specification mandates the usage of a single CNAME when sending RTP Streams in some circumstances, see Section 4.9.
- o Support for sending and receiving RTCP SR, RR, SDES, and BYE packet types. Note that support for other RTCP packet types is OPTIONAL, unless mandated by other parts of this specification. Note that additional RTCP Packet types are used by the RTP/SAVPF Profile (Section 4.2) and the other RTCP extensions (Section 5). WebRTC endpoints that implement the SDP bundle negotiation extension will use the SDP grouping framework 'mid' attribute to identify media streams. Such endpoints MUST implement the RTCP SDES MID item described in [I-D.ietf-mmusic-sdp-bundle-negotiation].
- o Support for multiple endpoints in a single RTP session, and for scaling the RTCP transmission interval according to the number of participants in the session; support for randomised RTCP transmission intervals to avoid synchronisation of RTCP reports; support for RTCP timer reconsideration (Section 6.3.6 of

[RFC3550]) and reverse reconsideration (Section 6.3.4 of [RFC3550]).

- o Support for configuring the RTCP bandwidth as a fraction of the media bandwidth, and for configuring the fraction of the RTCP bandwidth allocated to senders, e.g., using the SDP "b=" line [RFC4566][RFC3556].
- o Support for the reduced minimum RTCP reporting interval described in Section 6.2 of [RFC3550]. When using the reduced minimum RTCP reporting interval, the fixed (non-reduced) minimum interval MUST be used when calculating the participant timeout interval (see Sections 6.2 and 6.3.5 of [RFC3550]). The delay before sending the initial compound RTCP packet can be set to zero (see Section 6.2 of [RFC3550] as updated by [I-D.ietf-avtcore-rtp-multi-stream]).
- o Support for discontinuous transmission. RTP allows endpoints to pause and resume transmission at any time. When resuming, the RTP sequence number will increase by one, as usual, while the increase in the RTP timestamp value will depend on the duration of the pause. Discontinuous transmission is most commonly used with some audio payload formats, but is not audio specific, and can be used with any RTP payload format.
- o Ignore unknown RTCP packet types and RTP header extensions. This is to ensure robust handling of future extensions, middlebox behaviours, etc., that can result in not signalled RTCP packet types or RTP header extensions being received. If a compound RTCP packet is received that contains a mixture of known and unknown RTCP packet types, the known packets types need to be processed as usual, with only the unknown packet types being discarded.

It is known that a significant number of legacy RTP implementations, especially those targeted at VoIP-only systems, do not support all of the above features, and in some cases do not support RTCP at all. Implementers are advised to consider the requirements for graceful degradation when interoperating with legacy implementations.

Other implementation considerations are discussed in Section 12.

#### 4.2. Choice of the RTP Profile

The complete specification of RTP for a particular application domain requires the choice of an RTP Profile. For WebRTC use, the Extended Secure RTP Profile for RTCP-Based Feedback (RTP/SAVPF) [RFC5124], as extended by [RFC7007], MUST be implemented. The RTP/SAVPF profile is the combination of basic RTP/AVP profile [RFC3551], the RTP profile

for RTCP-based feedback (RTP/AVPF) [RFC4585], and the secure RTP profile (RTP/SAVP) [RFC3711].

The RTCP-based feedback extensions [RFC4585] are needed for the improved RTCP timer model. This allows more flexible transmission of RTCP packets in response to events, rather than strictly according to bandwidth, and is vital for being able to report congestion signals as well as media events. These extensions also allow saving RTCP bandwidth, and an endpoint will commonly only use the full RTCP bandwidth allocation if there are many events that require feedback. The timer rules are also needed to make use of the RTP conferencing extensions discussed in Section 5.1.

Note: The enhanced RTCP timer model defined in the RTP/AVPF profile is backwards compatible with legacy systems that implement only the RTP/AVP or RTP/SAVP profile, given some constraints on parameter configuration such as the RTCP bandwidth value and "trr-int" (the most important factor for interworking with RTP/(S)AVP endpoints via a gateway is to set the trr-int parameter to a value representing 4 seconds, see Section 6.1 in [I-D.ietf-avtcore-rtp-multi-stream]).

The secure RTP (SRTP) profile extensions [RFC3711] are needed to provide media encryption, integrity protection, replay protection and a limited form of source authentication. WebRTC Endpoints MUST NOT send packets using the basic RTP/AVP profile or the RTP/AVPF profile; they MUST employ the full RTP/SAVPF profile to protect all RTP and RTCP packets that are generated (i.e., implementations MUST use SRTP and SRTCP). The RTP/SAVPF profile MUST be configured using the cipher suites, DTLS-SRTP protection profiles, keying mechanisms, and other parameters described in [I-D.ietf-rtcweb-security-arch].

#### 4.3. Choice of RTP Payload Formats

Mandatory to implement audio codecs and RTP payload formats for WebRTC endpoints are defined in [I-D.ietf-rtcweb-audio]. Mandatory to implement video codecs and RTP payload formats for WebRTC endpoints are defined in [I-D.ietf-rtcweb-video]. WebRTC endpoints MAY additionally implement any other codec for which an RTP payload format and associated signalling has been defined.

WebRTC Endpoints cannot assume that the other participants in an RTP session understand any RTP payload format, no matter how common. The mapping between RTP payload type numbers and specific configurations of particular RTP payload formats MUST be agreed before those payload types/formats can be used. In an SDP context, this can be done using the "a=rtpmap:" and "a=fmtp:" attributes associated with an "m="

line, along with any other SDP attributes needed to configure the RTP payload format.

Endpoints can signal support for multiple RTP payload formats, or multiple configurations of a single RTP payload format, as long as each unique RTP payload format configuration uses a different RTP payload type number. As outlined in Section 4.8, the RTP payload type number is sometimes used to associate an RTP packet stream with a signalling context. This association is possible provided unique RTP payload type numbers are used in each context. For example, an RTP packet stream can be associated with an SDP "m=" line by comparing the RTP payload type numbers used by the RTP packet stream with payload types signalled in the "a=rtpmap:" lines in the media sections of the SDP. This leads to the following considerations:

If RTP packet streams are being associated with signalling contexts based on the RTP payload type, then the assignment of RTP payload type numbers MUST be unique across signalling contexts.

If the same RTP payload format configuration is used in multiple contexts, then a different RTP payload type number has to be assigned in each context to ensure uniqueness.

If the RTP payload type number is not being used to associate RTP packet streams with a signalling context, then the same RTP payload type number can be used to indicate the exact same RTP payload format configuration in multiple contexts.

A single RTP payload type number MUST NOT be assigned to different RTP payload formats, or different configurations of the same RTP payload format, within a single RTP session (note that the "m=" lines in an SDP bundle group [I-D.ietf-mmusic-sdp-bundle-negotiation] form a single RTP session).

An endpoint that has signalled support for multiple RTP payload formats MUST be able to accept data in any of those payload formats at any time, unless it has previously signalled limitations on its decoding capability. This requirement is constrained if several types of media (e.g., audio and video) are sent in the same RTP session. In such a case, a source (SSRC) is restricted to switching only between the RTP payload formats signalled for the type of media that is being sent by that source; see Section 4.4. To support rapid rate adaptation by changing codec, RTP does not require advance signalling for changes between RTP payload formats used by a single SSRC that were signalled during session set-up.

If performing changes between two RTP payload types that use different RTP clock rates, an RTP sender MUST follow the

recommendations in Section 4.1 of [RFC7160]. RTP receivers **MUST** follow the recommendations in Section 4.3 of [RFC7160] in order to support sources that switch between clock rates in an RTP session (these recommendations for receivers are backwards compatible with the case where senders use only a single clock rate).

#### 4.4. Use of RTP Sessions

An association amongst a set of endpoints communicating using RTP is known as an RTP session [RFC3550]. An endpoint can be involved in several RTP sessions at the same time. In a multimedia session, each type of media has typically been carried in a separate RTP session (e.g., using one RTP session for the audio, and a separate RTP session using a different transport-layer flow for the video). WebRTC Endpoints are **REQUIRED** to implement support for multimedia sessions in this way, separating each RTP session using different transport-layer flows for compatibility with legacy systems (this is sometimes called session multiplexing).

In modern day networks, however, with the widespread use of network address/port translators (NAT/NAPT) and firewalls, it is desirable to reduce the number of transport-layer flows used by RTP applications. This can be done by sending all the RTP packet streams in a single RTP session, which will comprise a single transport-layer flow (this will prevent the use of some quality-of-service mechanisms, as discussed in Section 12.1.3). Implementations are therefore also **REQUIRED** to support transport of all RTP packet streams, independent of media type, in a single RTP session using a single transport layer flow, according to [I-D.ietf-avtcore-multi-media-rtp-session] (this is sometimes called SSRC multiplexing). If multiple types of media are to be used in a single RTP session, all participants in that RTP session **MUST** agree to this usage. In an SDP context, [I-D.ietf-mmusic-sdp-bundle-negotiation] can be used to signal such a bundle of RTP packet streams forming a single RTP session.

Further discussion about the suitability of different RTP session structures and multiplexing methods to different scenarios can be found in [I-D.ietf-avtcore-multiplex-guidelines].

#### 4.5. RTP and RTCP Multiplexing

Historically, RTP and RTCP have been run on separate transport layer flows (e.g., two UDP ports for each RTP session, one port for RTP and one port for RTCP). With the increased use of Network Address/Port Translation (NAT/NAPT) this has become problematic, since maintaining multiple NAT bindings can be costly. It also complicates firewall administration, since multiple ports need to be opened to allow RTP traffic. To reduce these costs and session set-up times,

implementations are REQUIRED to support multiplexing RTP data packets and RTCP control packets on a single transport-layer flow [RFC5761]. Such RTP and RTCP multiplexing MUST be negotiated in the signalling channel before it is used. If SDP is used for signalling, this negotiation MUST use the mechanism defined in [RFC5761]. Implementations can also support sending RTP and RTCP on separate transport-layer flows, but this is OPTIONAL to implement. If an implementation does not support RTP and RTCP sent on separate transport-layer flows, it MUST indicate that using the mechanism defined in [I-D.ietf-mmusic-mux-exclusive].

Note that the use of RTP and RTCP multiplexed onto a single transport-layer flow ensures that there is occasional traffic sent on that port, even if there is no active media traffic. This can be useful to keep NAT bindings alive [RFC6263].

#### 4.6. Reduced Size RTCP

RTCP packets are usually sent as compound RTCP packets, and [RFC3550] requires that those compound packets start with an Sender Report (SR) or Receiver Report (RR) packet. When using frequent RTCP feedback messages under the RTP/AVPF Profile [RFC4585] these statistics are not needed in every packet, and unnecessarily increase the mean RTCP packet size. This can limit the frequency at which RTCP packets can be sent within the RTCP bandwidth share.

To avoid this problem, [RFC5506] specifies how to reduce the mean RTCP message size and allow for more frequent feedback. Frequent feedback, in turn, is essential to make real-time applications quickly aware of changing network conditions, and to allow them to adapt their transmission and encoding behaviour. Implementations MUST support sending and receiving non-compound RTCP feedback packets [RFC5506]. Use of non-compound RTCP packets MUST be negotiated using the signalling channel. If SDP is used for signalling, this negotiation MUST use the attributes defined in [RFC5506]. For backwards compatibility, implementations are also REQUIRED to support the use of compound RTCP feedback packets if the remote endpoint does not agree to the use of non-compound RTCP in the signalling exchange.

#### 4.7. Symmetric RTP/RTCP

To ease traversal of NAT and firewall devices, implementations are REQUIRED to implement and use Symmetric RTP [RFC4961]. The reason for using symmetric RTP is primarily to avoid issues with NATs and Firewalls by ensuring that the send and receive RTP packet streams, as well as RTCP, are actually bi-directional transport-layer flows. This will keep alive the NAT and firewall pinholes, and help indicate consent that the receive direction is a transport-layer flow the

intended recipient actually wants. In addition, it saves resources, specifically ports at the endpoints, but also in the network as NAT mappings or firewall state is not unnecessary bloated. The amount of per flow QoS state kept in the network is also reduced.

#### 4.8. Choice of RTP Synchronisation Source (SSRC)

Implementations are REQUIRED to support signalled RTP synchronisation source (SSRC) identifiers. If SDP is used, this MUST be done using the "a=ssrc:" SDP attribute defined in Section 4.1 and Section 5 of [RFC5576] and the "previous-ssrc" source attribute defined in Section 6.2 of [RFC5576]; other per-SSRC attributes defined in [RFC5576] MAY be supported.

While support for signalled SSRC identifiers is mandated, their use in an RTP session is OPTIONAL. Implementations MUST be prepared to accept RTP and RTCP packets using SSRCS that have not been explicitly signalled ahead of time. Implementations MUST support random SSRC assignment, and MUST support SSRC collision detection and resolution, according to [RFC3550]. When using signalled SSRC values, collision detection MUST be performed as described in Section 5 of [RFC5576].

It is often desirable to associate an RTP packet stream with a non-RTP context. For users of the WebRTC API a mapping between SSRCS and MediaStreamTracks are provided per Section 11. For gateways or other usages it is possible to associate an RTP packet stream with an "m=" line in a session description formatted using SDP. If SSRCS are signalled this is straightforward (in SDP the "a=ssrc:" line will be at the media level, allowing a direct association with an "m=" line). If SSRCS are not signalled, the RTP payload type numbers used in an RTP packet stream are often sufficient to associate that packet stream with a signalling context (e.g., if RTP payload type numbers are assigned as described in Section 4.3 of this memo, the RTP payload types used by an RTP packet stream can be compared with values in SDP "a=rtpmap:" lines, which are at the media level in SDP, and so map to an "m=" line).

#### 4.9. Generation of the RTCP Canonical Name (CNAME)

The RTCP Canonical Name (CNAME) provides a persistent transport-level identifier for an RTP endpoint. While the Synchronisation Source (SSRC) identifier for an RTP endpoint can change if a collision is detected, or when the RTP application is restarted, its RTCP CNAME is meant to stay unchanged for the duration of a RTCPeerConnection [W3C.WD-webrtc-20130910], so that RTP endpoints can be uniquely identified and associated with their RTP packet streams within a set of related RTP sessions.

Each RTP endpoint MUST have at least one RTCP CNAME, and that RTCP CNAME MUST be unique within the RTCPeerConnection. RTCP CNAMEs identify a particular synchronisation context, i.e., all SSRCs associated with a single RTCP CNAME share a common reference clock. If an endpoint has SSRCs that are associated with several unsynchronised reference clocks, and hence different synchronisation contexts, it will need to use multiple RTCP CNAMEs, one for each synchronisation context.

Taking the discussion in Section 11 into account, a WebRTC Endpoint MUST NOT use more than one RTCP CNAME in the RTP sessions belonging to single RTCPeerConnection (that is, an RTCPeerConnection forms a synchronisation context). RTP middleboxes MAY generate RTP packet streams associated with more than one RTCP CNAME, to allow them to avoid having to resynchronize media from multiple different endpoints part of a multi-party RTP session.

The RTP specification [RFC3550] includes guidelines for choosing a unique RTP CNAME, but these are not sufficient in the presence of NAT devices. In addition, long-term persistent identifiers can be problematic from a privacy viewpoint (Section 13). Accordingly, a WebRTC Endpoint MUST generate a new, unique, short-term persistent RTCP CNAME for each RTCPeerConnection, following [RFC7022], with a single exception; if explicitly requested at creation an RTCPeerConnection MAY use the same CNAME as an existing RTCPeerConnection within their common same-origin context.

An WebRTC Endpoint MUST support reception of any CNAME that matches the syntax limitations specified by the RTP specification [RFC3550] and cannot assume that any CNAME will be chosen according to the form suggested above.

#### 4.10. Handling of Leap Seconds

The guidelines regarding handling of leap seconds to limit their impact on RTP media play-out and synchronization given in [RFC7164] SHOULD be followed.

### 5. WebRTC Use of RTP: Extensions

There are a number of RTP extensions that are either needed to obtain full functionality, or extremely useful to improve on the baseline performance, in the WebRTC context. One set of these extensions is related to conferencing, while others are more generic in nature. The following subsections describe the various RTP extensions mandated or suggested for use within WebRTC.

### 5.1. Conferencing Extensions and Topologies

RTP is a protocol that inherently supports group communication. Groups can be implemented by having each endpoint send its RTP packet streams to an RTP middlebox that redistributes the traffic, by using a mesh of unicast RTP packet streams between endpoints, or by using an IP multicast group to distribute the RTP packet streams. These topologies can be implemented in a number of ways as discussed in [I-D.ietf-avtcore-rtp-topologies-update].

While the use of IP multicast groups is popular in IPTV systems, the topologies based on RTP middleboxes are dominant in interactive video conferencing environments. Topologies based on a mesh of unicast transport-layer flows to create a common RTP session have not seen widespread deployment to date. Accordingly, WebRTC Endpoints are not expected to support topologies based on IP multicast groups or to support mesh-based topologies, such as a point-to-multipoint mesh configured as a single RTP session (Topo-Mesh in the terminology of [I-D.ietf-avtcore-rtp-topologies-update]). However, a point-to-multipoint mesh constructed using several RTP sessions, implemented in WebRTC using independent `RTCPeerConnections` [W3C.WD-webrtc-20130910], can be expected to be used in WebRTC, and needs to be supported.

WebRTC Endpoints implemented according to this memo are expected to support all the topologies described in [I-D.ietf-avtcore-rtp-topologies-update] where the RTP endpoints send and receive unicast RTP packet streams to and from some peer device, provided that peer can participate in performing congestion control on the RTP packet streams. The peer device could be another RTP endpoint, or it could be an RTP middlebox that redistributes the RTP packet streams to other RTP endpoints. This limitation means that some of the RTP middlebox-based topologies are not suitable for use in WebRTC. Specifically:

- o Video switching MCUs (Topo-Video-switch-MCU) SHOULD NOT be used, since they make the use of RTCP for congestion control and quality of service reports problematic (see Section 3.8 of [I-D.ietf-avtcore-rtp-topologies-update]).
- o The Relay-Transport Translator (Topo-PtM-Trn-Translator) topology SHOULD NOT be used because its safe use requires a congestion control algorithm or RTP circuit breaker that handles point to multipoint, which has not yet been standardised.

The following topology can be used, however it has some issues worth noting:

- o Content modifying MCUs with RTCP termination (Topo-RTCP-terminating-MCU) MAY be used. Note that in this RTP Topology, RTP loop detection and identification of active senders is the responsibility of the WebRTC application; since the clients are isolated from each other at the RTP layer, RTP cannot assist with these functions (see section 3.9 of [I-D.ietf-avtcore-rtp-topologies-update]).

The RTP extensions described in Section 5.1.1 to Section 5.1.6 are designed to be used with centralised conferencing, where an RTP middlebox (e.g., a conference bridge) receives a participant's RTP packet streams and distributes them to the other participants. These extensions are not necessary for interoperability; an RTP endpoint that does not implement these extensions will work correctly, but might offer poor performance. Support for the listed extensions will greatly improve the quality of experience and, to provide a reasonable baseline quality, some of these extensions are mandatory to be supported by WebRTC Endpoints.

The RTCP conferencing extensions are defined in Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF) [RFC4585] and the memo on Codec Control Messages (CCM) in RTP/AVPF [RFC5104]; they are fully usable by the Secure variant of this profile (RTP/SAVPF) [RFC5124].

#### 5.1.1. Full Intra Request (FIR)

The Full Intra Request message is defined in Sections 3.5.1 and 4.3.1 of the Codec Control Messages [RFC5104]. It is used to make the mixer request a new Intra picture from a participant in the session. This is used when switching between sources to ensure that the receivers can decode the video or other predictive media encoding with long prediction chains. WebRTC Endpoints that are sending media MUST understand and react to FIR feedback messages they receive, since this greatly improves the user experience when using centralised mixer-based conferencing. Support for sending FIR messages is OPTIONAL.

#### 5.1.2. Picture Loss Indication (PLI)

The Picture Loss Indication message is defined in Section 6.3.1 of the RTP/AVPF profile [RFC4585]. It is used by a receiver to tell the sending encoder that it lost the decoder context and would like to have it repaired somehow. This is semantically different from the Full Intra Request above as there could be multiple ways to fulfil the request. WebRTC Endpoints that are sending media MUST understand and react to PLI feedback messages as a loss tolerance mechanism. Receivers MAY send PLI messages.

#### 5.1.3. Slice Loss Indication (SLI)

The Slice Loss Indication message is defined in Section 6.3.2 of the RTP/AVPF profile [RFC4585]. It is used by a receiver to tell the encoder that it has detected the loss or corruption of one or more consecutive macro blocks, and would like to have these repaired somehow. It is RECOMMENDED that receivers generate SLI feedback messages if slices are lost when using a codec that supports the concept of macro blocks. A sender that receives an SLI feedback message SHOULD attempt to repair the lost slice(s).

#### 5.1.4. Reference Picture Selection Indication (RPSI)

Reference Picture Selection Indication (RPSI) messages are defined in Section 6.3.3 of the RTP/AVPF profile [RFC4585]. Some video encoding standards allow the use of older reference pictures than the most recent one for predictive coding. If such a codec is in use, and if the encoder has learnt that encoder-decoder synchronisation has been lost, then a known as correct reference picture can be used as a base for future coding. The RPSI message allows this to be signalled. Receivers that detect that encoder-decoder synchronisation has been lost SHOULD generate an RPSI feedback message if codec being used supports reference picture selection. A RTP packet stream sender that receives such an RPSI message SHOULD act on that messages to change the reference picture, if it is possible to do so within the available bandwidth constraints, and with the codec being used.

#### 5.1.5. Temporal-Spatial Trade-off Request (TSTR)

The temporal-spatial trade-off request and notification are defined in Sections 3.5.2 and 4.3.2 of [RFC5104]. This request can be used to ask the video encoder to change the trade-off it makes between temporal and spatial resolution, for example to prefer high spatial image quality but low frame rate. Support for TSTR requests and notifications is OPTIONAL.

#### 5.1.6. Temporary Maximum Media Stream Bit Rate Request (TMMBR)

The TMMBR feedback message is defined in Sections 3.5.4 and 4.2.1 of the Codec Control Messages [RFC5104]. This request and its notification message are used by a media receiver to inform the sending party that there is a current limitation on the amount of bandwidth available to this receiver. There can be various reasons for this: for example, an RTP mixer can use this message to limit the media rate of the sender being forwarded by the mixer (without doing media transcoding) to fit the bottlenecks existing towards the other session participants. WebRTC Endpoints that are sending media are REQUIRED to implement support for TMMBR messages, and MUST follow

bandwidth limitations set by a TMMBR message received for their SSRC. The sending of TMMBR requests is OPTIONAL.

## 5.2. Header Extensions

The RTP specification [RFC3550] provides the capability to include RTP header extensions containing in-band data, but the format and semantics of the extensions are poorly specified. The use of header extensions is OPTIONAL in WebRTC, but if they are used, they MUST be formatted and signalled following the general mechanism for RTP header extensions defined in [RFC5285], since this gives well-defined semantics to RTP header extensions.

As noted in [RFC5285], the requirement from the RTP specification that header extensions are "designed so that the header extension may be ignored" [RFC3550] stands. To be specific, header extensions MUST only be used for data that can safely be ignored by the recipient without affecting interoperability, and MUST NOT be used when the presence of the extension has changed the form or nature of the rest of the packet in a way that is not compatible with the way the stream is signalled (e.g., as defined by the payload type). Valid examples of RTP header extensions might include metadata that is additional to the usual RTP information, but that can safely be ignored without compromising interoperability.

### 5.2.1. Rapid Synchronisation

Many RTP sessions require synchronisation between audio, video, and other content. This synchronisation is performed by receivers, using information contained in RTCP SR packets, as described in the RTP specification [RFC3550]. This basic mechanism can be slow, however, so it is RECOMMENDED that the rapid RTP synchronisation extensions described in [RFC6051] be implemented in addition to RTCP SR-based synchronisation.

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

### 5.2.2. Client-to-Mixer Audio Level

The Client to Mixer Audio Level extension [RFC6464] is an RTP header extension used by an endpoint to inform a mixer about the level of audio activity in the packet to which the header is attached. This enables an RTP middlebox to make mixing or selection decisions without decoding or detailed inspection of the payload, reducing the complexity in some types of mixers. It can also save decoding resources in receivers, which can choose to decode only the most relevant RTP packet streams based on audio activity levels.

The Client-to-Mixer Audio Level [RFC6464] header extension MUST be implemented. It is REQUIRED that implementations are capable of encrypting the header extension according to [RFC6904] since the information contained in these header extensions can be considered sensitive. The use of this encryption is RECOMMENDED, however usage of the encryption can be explicitly disabled through API or signalling.

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

#### 5.2.3. Mixer-to-Client Audio Level

The Mixer to Client Audio Level header extension [RFC6465] provides an endpoint with the audio level of the different sources mixed into a common source stream by a RTP mixer. This enables a user interface to indicate the relative activity level of each session participant, rather than just being included or not based on the CSRC field. This is a pure optimisation of non critical functions, and is hence OPTIONAL to implement. If this header extension is implemented, it is REQUIRED that implementations are capable of encrypting the header extension according to [RFC6904] since the information contained in these header extensions can be considered sensitive. It is further RECOMMENDED that this encryption is used, unless the encryption has been explicitly disabled through API or signalling.

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

#### 5.2.4. Media Stream Identification

WebRTC endpoints that implement the SDP bundle negotiation extension will use the SDP grouping framework 'mid' attribute to identify media streams. Such endpoints MUST implement the RTP MID header extension described in [I-D.ietf-mmusic-sdp-bundle-negotiation].

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

#### 5.2.5. Coordination of Video Orientation

WebRTC endpoints that send or receive video MUST implement the coordination of video orientation (CVO) RTP header extension as described in Section 4 of [I-D.ietf-rtcweb-video].

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

## 6. WebRTC Use of RTP: Improving Transport Robustness

There are tools that can make RTP packet streams robust against packet loss and reduce the impact of loss on media quality. However, they generally add some overhead compared to a non-robust stream. The overhead needs to be considered, and the aggregate bit-rate **MUST** be rate controlled to avoid causing network congestion (see Section 7). As a result, improving robustness might require a lower base encoding quality, but has the potential to deliver that quality with fewer errors. The mechanisms described in the following sub-sections can be used to improve tolerance to packet loss.

### 6.1. Negative Acknowledgements and RTP Retransmission

As a consequence of supporting the RTP/SAVPF profile, implementations can send negative acknowledgements (NACKs) for RTP data packets [RFC4585]. This feedback can be used to inform a sender of the loss of particular RTP packets, subject to the capacity limitations of the RTCP feedback channel. A sender can use this information to optimise the user experience by adapting the media encoding to compensate for known lost packets.

RTP packet stream senders are **REQUIRED** to understand the Generic NACK message defined in Section 6.2.1 of [RFC4585], but **MAY** choose to ignore some or all of this feedback (following Section 4.2 of [RFC4585]). Receivers **MAY** send NACKs for missing RTP packets. Guidelines on when to send NACKs are provided in [RFC4585]. It is not expected that a receiver will send a NACK for every lost RTP packet, rather it needs to consider the cost of sending NACK feedback, and the importance of the lost packet, to make an informed decision on whether it is worth telling the sender about a packet loss event.

The RTP Retransmission Payload Format [RFC4588] offers the ability to retransmit lost packets based on NACK feedback. Retransmission needs to be used with care in interactive real-time applications to ensure that the retransmitted packet arrives in time to be useful, but can be effective in environments with relatively low network RTT (an RTP sender can estimate the RTT to the receivers using the information in RTCP SR and RR packets, as described at the end of Section 6.4.1 of [RFC3550]). The use of retransmissions can also increase the forward RTP bandwidth, and can potentially caused increased packet loss if the original packet loss was caused by network congestion. Note, however, that retransmission of an important lost packet to repair decoder state can have lower cost than sending a full intra frame. It is not appropriate to blindly retransmit RTP packets in response to a NACK. The importance of lost packets and the likelihood of them

arriving in time to be useful needs to be considered before RTP retransmission is used.

Receivers are REQUIRED to implement support for RTP retransmission packets [RFC4588] sent using SSRC multiplexing, and MAY also support RTP retransmission packets sent using session multiplexing. Senders MAY send RTP retransmission packets in response to NACKs if support for the RTP retransmission payload format has been negotiated, and if the sender believes it is useful to send a retransmission of the packet(s) referenced in the NACK. Senders do not need to retransmit every NACKed packet.

## 6.2. Forward Error Correction (FEC)

The use of Forward Error Correction (FEC) can provide an effective protection against some degree of packet loss, at the cost of steady bandwidth overhead. There are several FEC schemes that are defined for use with RTP. Some of these schemes are specific to a particular RTP payload format, others operate across RTP packets and can be used with any payload format. It needs to be noted that using redundant encoding or FEC will lead to increased play out delay, which needs to be considered when choosing FEC schemes and their parameters.

WebRTC endpoints MUST follow the recommendations for FEC use given in [I-D.ietf-rtcweb-fec]. WebRTC endpoints MAY support other types of FEC, but these MUST be negotiated before they are used.

## 7. WebRTC Use of RTP: Rate Control and Media Adaptation

WebRTC will be used in heterogeneous network environments using a variety of link technologies, including both wired and wireless links, to interconnect potentially large groups of users around the world. As a result, the network paths between users can have widely varying one-way delays, available bit-rates, load levels, and traffic mixtures. Individual endpoints can send one or more RTP packet streams to each participant, and there can be several participants. Each of these RTP packet streams can contain different types of media, and the type of media, bit rate, and number of RTP packet streams as well as transport-layer flows can be highly asymmetric. Non-RTP traffic can share the network paths with RTP transport-layer flows. Since the network environment is not predictable or stable, WebRTC Endpoints MUST ensure that the RTP traffic they generate can adapt to match changes in the available network capacity.

The quality of experience for users of WebRTC is very dependent on effective adaptation of the media to the limitations of the network. Endpoints have to be designed so they do not transmit significantly more data than the network path can support, except for very short

time periods, otherwise high levels of network packet loss or delay spikes will occur, causing media quality degradation. The limiting factor on the capacity of the network path might be the link bandwidth, or it might be competition with other traffic on the link (this can be non-WebRTC traffic, traffic due to other WebRTC flows, or even competition with other WebRTC flows in the same session).

An effective media congestion control algorithm is therefore an essential part of the WebRTC framework. However, at the time of this writing, there is no standard congestion control algorithm that can be used for interactive media applications such as WebRTC's flows. Some requirements for congestion control algorithms for RTCPeerConnections are discussed in [I-D.ietf-rmcat-cc-requirements]. If a standardized congestion control algorithm that satisfies these requirements is developed in the future, this memo will need to be updated to mandate its use.

#### 7.1. Boundary Conditions and Circuit Breakers

WebRTC Endpoints MUST implement the RTP circuit breaker algorithm that is described in [I-D.ietf-avtcore-rtp-circuit-breakers]. The RTP circuit breaker is designed to enable applications to recognise and react to situations of extreme network congestion. However, since the RTP circuit breaker might not be triggered until congestion becomes extreme, it cannot be considered a substitute for congestion control, and applications MUST also implement congestion control to allow them to adapt to changes in network capacity. The congestion control algorithm will have to be proprietary until a standardized congestion control algorithm is available. Any future RTP congestion control algorithms are expected to operate within the envelope allowed by the circuit breaker.

The session establishment signalling will also necessarily establish boundaries to which the media bit-rate will conform. The choice of media codecs provides upper- and lower-bounds on the supported bit-rates that the application can utilise to provide useful quality, and the packetisation choices that exist. In addition, the signalling channel can establish maximum media bit-rate boundaries using, for example, the SDP "b=AS:" or "b=CT:" lines and the RTP/AVPF Temporary Maximum Media Stream Bit Rate (TMMBR) Requests (see Section 5.1.6 of this memo). Signalled bandwidth limitations, such as SDP "b=AS:" or "b=CT:" lines received from the peer, MUST be followed when sending RTP packet streams. A WebRTC Endpoint receiving media SHOULD signal its bandwidth limitations. These limitations have to be based on known bandwidth limitations, for example the capacity of the edge links.

## 7.2. Congestion Control Interoperability and Legacy Systems

All endpoints that wish to interwork with WebRTC MUST implement RTCP and provide congestion feedback via the defined RTCP reporting mechanisms.

When interworking with legacy implementations that support RTCP using the RTP/AVP profile [RFC3551], congestion feedback is provided in RTCP RR packets every few seconds. Implementations that have to interwork with such endpoints MUST ensure that they keep within the RTP circuit breaker [I-D.ietf-avtcore-rtp-circuit-breakers] constraints to limit the congestion they can cause.

If a legacy endpoint supports RTP/AVPF, this enables negotiation of important parameters for frequent reporting, such as the "trr-int" parameter, and the possibility that the endpoint supports some useful feedback format for congestion control purpose such as TMMBR [RFC5104]. Implementations that have to interwork with such endpoints MUST ensure that they stay within the RTP circuit breaker [I-D.ietf-avtcore-rtp-circuit-breakers] constraints to limit the congestion they can cause, but might find that they can achieve better congestion response depending on the amount of feedback that is available.

With proprietary congestion control algorithms issues can arise when different algorithms and implementations interact in a communication session. If the different implementations have made different choices in regards to the type of adaptation, for example one sender based, and one receiver based, then one could end up in situation where one direction is dual controlled, when the other direction is not controlled. This memo cannot mandate behaviour for proprietary congestion control algorithms, but implementations that use such algorithms ought to be aware of this issue, and try to ensure that effective congestion control is negotiated for media flowing in both directions. If the IETF were to standardise both sender- and receiver-based congestion control algorithms for WebRTC traffic in the future, the issues of interoperability, control, and ensuring that both directions of media flow are congestion controlled would also need to be considered.

## 8. WebRTC Use of RTP: Performance Monitoring

As described in Section 4.1, implementations are REQUIRED to generate RTCP Sender Report (SR) and Reception Report (RR) packets relating to the RTP packet streams they send and receive. These RTCP reports can be used for performance monitoring purposes, since they include basic packet loss and jitter statistics.

A large number of additional performance metrics are supported by the RTCP Extended Reports (XR) framework, see [RFC3611][RFC6792]. At the time of this writing, it is not clear what extended metrics are suitable for use in WebRTC, so there is no requirement that implementations generate RTCP XR packets. However, implementations that can use detailed performance monitoring data MAY generate RTCP XR packets as appropriate. The use of RTCP XR packets SHOULD be signalled; implementations MUST ignore RTCP XR packets that are unexpected or not understood.

#### 9. WebRTC Use of RTP: Future Extensions

It is possible that the core set of RTP protocols and RTP extensions specified in this memo will prove insufficient for the future needs of WebRTC. In this case, future updates to this memo have to be made following the Guidelines for Writers of RTP Payload Format Specifications [RFC2736], How to Write an RTP Payload Format [I-D.ietf-payload-rtp-howto] and Guidelines for Extending the RTP Control Protocol [RFC5968], and SHOULD take into account any future guidelines for extending RTP and related protocols that have been developed.

Authors of future extensions are urged to consider the wide range of environments in which RTP is used when recommending extensions, since extensions that are applicable in some scenarios can be problematic in others. Where possible, the WebRTC framework will adopt RTP extensions that are of general utility, to enable easy implementation of a gateway to other applications using RTP, rather than adopt mechanisms that are narrowly targeted at specific WebRTC use cases.

#### 10. Signalling Considerations

RTP is built with the assumption that an external signalling channel exists, and can be used to configure RTP sessions and their features. The basic configuration of an RTP session consists of the following parameters:

RTP Profile: The name of the RTP profile to be used in session. The RTP/AVP [RFC3551] and RTP/AVPF [RFC4585] profiles can interoperate on basic level, as can their secure variants RTP/SAVP [RFC3711] and RTP/SAVPF [RFC5124]. The secure variants of the profiles do not directly interoperate with the non-secure variants, due to the presence of additional header fields for authentication in SRTP packets and cryptographic transformation of the payload. WebRTC requires the use of the RTP/SAVPF profile, and this MUST be signalled. Interworking functions might transform this into the RTP/SAVP profile for a legacy use case, by indicating to the

WebRTC Endpoint that the RTP/SAVPF is used and configuring a `trr-int` value of 4 seconds.

**Transport Information:** Source and destination IP address(s) and ports for RTP and RTCP MUST be signalled for each RTP session. In WebRTC these transport addresses will be provided by ICE [RFC5245] that signals candidates and arrives at nominated candidate address pairs. If RTP and RTCP multiplexing [RFC5761] is to be used, such that a single port, i.e. transport-layer flow, is used for RTP and RTCP flows, this MUST be signalled (see Section 4.5).

**RTP Payload Types, media formats, and format parameters:** The mapping between media type names (and hence the RTP payload formats to be used), and the RTP payload type numbers MUST be signalled. Each media type MAY also have a number of media type parameters that MUST also be signalled to configure the codec and RTP payload format (the "a=fmtp:" line from SDP). Section 4.3 of this memo discusses requirements for uniqueness of payload types.

**RTP Extensions:** The use of any additional RTP header extensions and RTCP packet types, including any necessary parameters, MUST be signalled. This signalling is to ensure that a WebRTC Endpoint's behaviour, especially when sending, of any extensions is predictable and consistent. For robustness, and for compatibility with non-WebRTC systems that might be connected to a WebRTC session via a gateway, implementations are REQUIRED to ignore unknown RTCP packets and RTP header extensions (see also Section 4.1).

**RTCP Bandwidth:** Support for exchanging RTCP Bandwidth values to the endpoints will be necessary. This SHALL be done as described in "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth" [RFC3556] if using SDP, or something semantically equivalent. This also ensures that the endpoints have a common view of the RTCP bandwidth. A common view of the RTCP bandwidth among different endpoints is important, to prevent differences in RTCP packet timing and timeout intervals causing interoperability problems.

These parameters are often expressed in SDP messages conveyed within an offer/answer exchange. RTP does not depend on SDP or on the offer/answer model, but does require all the necessary parameters to be agreed upon, and provided to the RTP implementation. Note that in WebRTC it will depend on the signalling model and API how these parameters need to be configured but they will be need to either be set in the API or explicitly signalled between the peers.

## 11. WebRTC API Considerations

The WebRTC API [W3C.WD-webrtc-20130910] and the Media Capture and Streams API [W3C.WD-mediacapture-streams-20130903] defines and uses the concept of a `MediaStream` that consists of zero or more `MediaStreamTracks`. A `MediaStreamTrack` is an individual stream of media from any type of media source like a microphone or a camera, but also conceptual sources, like a audio mix or a video composition, are possible. The `MediaStreamTracks` within a `MediaStream` might need to be synchronized during play back.

A `MediaStreamTrack`'s realisation in RTP in the context of an `RTCPeerConnection` consists of a source packet stream identified with an SSRC within an RTP session part of the `RTCPeerConnection`. The `MediaStreamTrack` can also result in additional packet streams, and thus SSRCs, in the same RTP session. These can be dependent packet streams from scalable encoding of the source stream associated with the `MediaStreamTrack`, if such a media encoder is used. They can also be redundancy packet streams, these are created when applying Forward Error Correction (Section 6.2) or RTP retransmission (Section 6.1) to the source packet stream.

It is important to note that the same media source can be feeding multiple `MediaStreamTracks`. As different sets of constraints or other parameters can be applied to the `MediaStreamTrack`, each `MediaStreamTrack` instance added to a `RTCPeerConnection` SHALL result in an independent source packet stream, with its own set of associated packet streams, and thus different SSRC(s). It will depend on applied constraints and parameters if the source stream and the encoding configuration will be identical between different `MediaStreamTracks` sharing the same media source. If the encoding parameters and constraints are the same, an implementation could choose to use only one encoded stream to create the different RTP packet streams. Note that such optimisations would need to take into account that the constraints for one of the `MediaStreamTracks` can at any moment change, meaning that the encoding configurations might no longer be identical and two different encoder instances would then be needed.

The same `MediaStreamTrack` can also be included in multiple `MediaStreams`, thus multiple sets of `MediaStreams` can implicitly need to use the same synchronisation base. To ensure that this works in all cases, and does not force an endpoint to disrupt the media by changing synchronisation base and CNAME during delivery of any ongoing packet streams, all `MediaStreamTracks` and their associated SSRCs originating from the same endpoint need to be sent using the same CNAME within one `RTCPeerConnection`. This is motivating the use of a single CNAME in Section 4.9.

The requirement on using the same CNAME for all SSRCs that originate from the same endpoint, does not require a middlebox that forwards traffic from multiple endpoints to only use a single CNAME.

Different CNAMEs normally need to be used for different `RTCPeerConnection` instances, as specified in Section 4.9. Having two communication sessions with the same CNAME could enable tracking of a user or device across different services (see Section 4.4.1 of [I-D.ietf-rtcweb-security] for details). A web application can request that the CNAMEs used in different `RTCPeerConnections` (within a same-origin context) be the same, this allows for synchronization of the endpoint's RTP packet streams across the different `RTCPeerConnections`.

Note: this doesn't result in a tracking issue, since the creation of matching CNAMEs depends on existing tracking within a single origin.

The above will currently force a WebRTC Endpoint that receives a `MediaStreamTrack` on one `RTCPeerConnection` and adds it as an outgoing on any `RTCPeerConnection` to perform resynchronisation of the stream. Since the sending party needs to change the CNAME to the one it uses, this implies it has to use a local system clock as timebase for the synchronisation. Thus, the relative relation between the timebase of the incoming stream and the system sending out needs to be defined. This relation also needs monitoring for clock drift and likely adjustments of the synchronisation. The sending entity is also responsible for congestion control for its sent streams. In cases of packet loss the loss of incoming data also needs to be handled. This leads to the observation that the method that is least likely to cause issues or interruptions in the outgoing source packet stream is a model of full decoding, including repair etc., followed by encoding of the media again into the outgoing packet stream. Optimisations of this method are clearly possible and implementation specific.

A WebRTC Endpoint MUST support receiving multiple `MediaStreamTracks`, where each of the different `MediaStreamTracks` (and their sets of associated packet streams) uses different CNAMEs. However, `MediaStreamTracks` that are received with different CNAMEs have no defined synchronisation.

Note: The motivation for supporting reception of multiple CNAMEs is to allow for forward compatibility with any future changes that enable more efficient stream handling when endpoints relay/forward streams. It also ensures that endpoints can interoperate with certain types of multi-stream middleboxes or endpoints that are not WebRTC.

Javascript Session Establishment Protocol [I-D.ietf-rtcweb-jsep] specifies that the binding between the WebRTC MediaStreams, MediaStreamTracks and the SSRC is done as specified in "Cross Session Stream Identification in the Session Description Protocol" [I-D.ietf-mmusic-msid]. The MSID document [I-D.ietf-mmusic-msid] also defines, in section 4.1, how to map unknown source packet stream SSRCs to MediaStreamTracks and MediaStreams. This later is relevant to handle some cases of legacy interoperability. Commonly the RTP Payload Type of any incoming packets will reveal if the packet stream is a source stream or a redundancy or dependent packet stream. The association to the correct source packet stream depends on the payload format in use for the packet stream.

Finally this specification puts a requirement on the WebRTC API to realize a method for determining the CSRC list (Section 4.1) as well as the Mixer-to-Client audio levels (Section 5.2.3) (when supported) and the basic requirements for this is further discussed in Section 12.2.1.

## 12. RTP Implementation Considerations

The following discussion provides some guidance on the implementation of the RTP features described in this memo. The focus is on a WebRTC Endpoint implementation perspective, and while some mention is made of the behaviour of middleboxes, that is not the focus of this memo.

### 12.1. Configuration and Use of RTP Sessions

A WebRTC Endpoint will be a simultaneous participant in one or more RTP sessions. Each RTP session can convey multiple media sources, and can include media data from multiple endpoints. In the following, some ways in which WebRTC Endpoints can configure and use RTP sessions are outlined.

#### 12.1.1. Use of Multiple Media Sources Within an RTP Session

RTP is a group communication protocol, and every RTP session can potentially contain multiple RTP packet streams. There are several reasons why this might be desirable:

Multiple media types: Outside of WebRTC, it is common to use one RTP session for each type of media source (e.g., one RTP session for audio sources and one for video sources, each sent over different transport layer flows). However, to reduce the number of UDP ports used, the default in WebRTC is to send all types of media in a single RTP session, as described in Section 4.4, using RTP and RTCP multiplexing (Section 4.5) to further reduce the number of UDP ports needed. This RTP session then uses only one bi-

directional transport-layer flow, but will contain multiple RTP packet streams, each containing a different type of media. A common example might be an endpoint with a camera and microphone that sends two RTP packet streams, one video and one audio, into a single RTP session.

**Multiple Capture Devices:** A WebRTC Endpoint might have multiple cameras, microphones, or other media capture devices, and so might want to generate several RTP packet streams of the same media type. Alternatively, it might want to send media from a single capture device in several different formats or quality settings at once. Both can result in a single endpoint sending multiple RTP packet streams of the same media type into a single RTP session at the same time.

**Associated Repair Data:** An endpoint might send a RTP packet stream that is somehow associated with another stream. For example, it might send an RTP packet stream that contains FEC or retransmission data relating to another stream. Some RTP payload formats send this sort of associated repair data as part of the source packet stream, while others send it as a separate packet stream.

**Layered or Multiple Description Coding:** An endpoint can use a layered media codec, for example H.264 SVC, or a multiple description codec, that generates multiple RTP packet streams, each with a distinct RTP SSRC, within a single RTP session.

**RTP Mixers, Translators, and Other Middleboxes:** An RTP session, in the WebRTC context, is a point-to-point association between an endpoint and some other peer device, where those devices share a common SSRC space. The peer device might be another WebRTC Endpoint, or it might be an RTP mixer, translator, or some other form of media processing middlebox. In the latter cases, the middlebox might send mixed or relayed RTP streams from several participants, that the WebRTC Endpoint will need to render. Thus, even though a WebRTC Endpoint might only be a member of a single RTP session, the peer device might be extending that RTP session to incorporate other endpoints. WebRTC is a group communication environment and endpoints need to be capable of receiving, decoding, and playing out multiple RTP packet streams at once, even in a single RTP session.

#### 12.1.2. Use of Multiple RTP Sessions

In addition to sending and receiving multiple RTP packet streams within a single RTP session, a WebRTC Endpoint might participate in

multiple RTP sessions. There are several reasons why a WebRTC Endpoint might choose to do this:

To interoperate with legacy devices: The common practice in the non-WebRTC world is to send different types of media in separate RTP sessions, for example using one RTP session for audio and another RTP session, on a separate transport layer flow, for video. All WebRTC Endpoints need to support the option of sending different types of media on different RTP sessions, so they can interwork with such legacy devices. This is discussed further in Section 4.4.

To provide enhanced quality of service: Some network-based quality of service mechanisms operate on the granularity of transport layer flows. If it is desired to use these mechanisms to provide differentiated quality of service for some RTP packet streams, then those RTP packet streams need to be sent in a separate RTP session using a different transport-layer flow, and with appropriate quality of service marking. This is discussed further in Section 12.1.3.

To separate media with different purposes: An endpoint might want to send RTP packet streams that have different purposes on different RTP sessions, to make it easy for the peer device to distinguish them. For example, some centralised multiparty conferencing systems display the active speaker in high resolution, but show low resolution "thumbnails" of other participants. Such systems might configure the endpoints to send simulcast high- and low-resolution versions of their video using separate RTP sessions, to simplify the operation of the RTP middlebox. In the WebRTC context this is currently possible by establishing multiple WebRTC MediaStreamTracks that have the same media source in one (or more) RTCPeerConnection. Each MediaStreamTrack is then configured to deliver a particular media quality and thus media bit-rate, and will produce an independently encoded version with the codec parameters agreed specifically in the context of that RTCPeerConnection. The RTP middlebox can distinguish packets corresponding to the low- and high-resolution streams by inspecting their SSRC, RTP payload type, or some other information contained in RTP payload, RTP header extension or RTCP packets, but it can be easier to distinguish the RTP packet streams if they arrive on separate RTP sessions on separate transport-layer flows.

To directly connect with multiple peers: A multi-party conference does not need to use an RTP middlebox. Rather, a multi-unicast mesh can be created, comprising several distinct RTP sessions, with each participant sending RTP traffic over a separate RTP session (that is, using an independent RTCPeerConnection object)

to every other participant, as shown in Figure 1. This topology has the benefit of not requiring an RTP middlebox node that is trusted to access and manipulate the media data. The downside is that it increases the used bandwidth at each sender by requiring one copy of the RTP packet streams for each participant that are part of the same session beyond the sender itself.

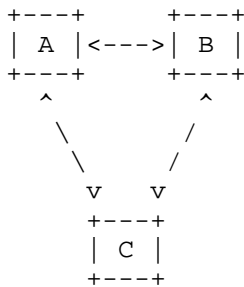


Figure 1: Multi-unicast using several RTP sessions

The multi-unicast topology could also be implemented as a single RTP session, spanning multiple peer-to-peer transport layer connections, or as several pairwise RTP sessions, one between each pair of peers. To maintain a coherent mapping of the relationship between RTP sessions and `RTCPeerConnection` objects it is recommended that this is implemented as several individual RTP sessions. The only downside is that endpoint A will not learn of the quality of any transmission happening between B and C, since it will not see RTCP reports for the RTP session between B and C, whereas it would if all three participants were part of a single RTP session. Experience with the Mbone tools (experimental RTP-based multicast conferencing tools from the late 1990s) has showed that RTCP reception quality reports for third parties can be presented to users in a way that helps them understand asymmetric network problems, and the approach of using separate RTP sessions prevents this. However, an advantage of using separate RTP sessions is that it enables using different media bit-rates and RTP session configurations between the different peers, thus not forcing B to endure the same quality reductions if there are limitations in the transport from A to C as C will. It is believed that these advantages outweigh the limitations in debugging power.

To indirectly connect with multiple peers: A common scenario in multi-party conferencing is to create indirect connections to multiple peers, using an RTP mixer, translator, or some other type of RTP middlebox. Figure 2 outlines a simple topology that might be used in a four-person centralised conference. The middlebox

acts to optimise the transmission of RTP packet streams from certain perspectives, either by only sending some of the received RTP packet stream to any given receiver, or by providing a combined RTP packet stream out of a set of contributing streams.

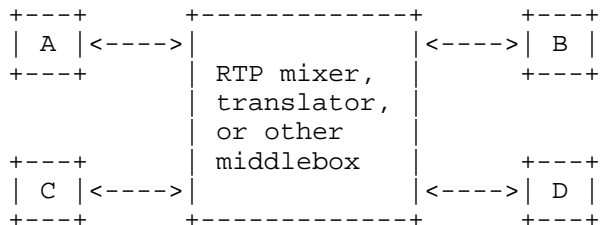


Figure 2: RTP mixer with only unicast paths

There are various methods of implementation for the middlebox. If implemented as a standard RTP mixer or translator, a single RTP session will extend across the middlebox and encompass all the endpoints in one multi-party session. Other types of middlebox might use separate RTP sessions between each endpoint and the middlebox. A common aspect is that these RTP middleboxes can use a number of tools to control the media encoding provided by a WebRTC Endpoint. This includes functions like requesting the breaking of the encoding chain and have the encoder produce a so called Intra frame. Another is limiting the bit-rate of a given stream to better suit the mixer view of the multiple down-streams. Others are controlling the most suitable frame-rate, picture resolution, the trade-off between frame-rate and spatial quality. The middlebox has the responsibility to correctly perform congestion control, source identification, manage synchronisation while providing the application with suitable media optimisations. The middlebox also has to be a trusted node when it comes to security, since it manipulates either the RTP header or the media itself (or both) received from one endpoint, before sending it on towards the endpoint(s), thus they need to be able to decrypt and then re-encrypt the RTP packet stream before sending it out.

RTP Mixers can create a situation where an endpoint experiences a situation in-between a session with only two endpoints and multiple RTP sessions. Mixers are expected to not forward RTCP reports regarding RTP packet streams across themselves. This is due to the difference in the RTP packet streams provided to the different endpoints. The original media source lacks information about a mixer's manipulations prior to sending it the different receivers. This scenario also results in that an endpoint's

feedback or requests go to the mixer. When the mixer can't act on this by itself, it is forced to go to the original media source to fulfil the receivers request. This will not necessarily be explicitly visible to any RTP and RTCP traffic, but the interactions and the time to complete them will indicate such dependencies.

Providing source authentication in multi-party scenarios is a challenge. In the mixer-based topologies, endpoints source authentication is based on, firstly, verifying that media comes from the mixer by cryptographic verification and, secondly, trust in the mixer to correctly identify any source towards the endpoint. In RTP sessions where multiple endpoints are directly visible to an endpoint, all endpoints will have knowledge about each others' master keys, and can thus inject packets claimed to come from another endpoint in the session. Any node performing relay can perform non-cryptographic mitigation by preventing forwarding of packets that have SSRC fields that came from other endpoints before. For cryptographic verification of the source, SRTP would require additional security mechanisms, for example TESLA for SRTP [RFC4383], that are not part of the base WebRTC standards.

To forward media between multiple peers: It is sometimes desirable for an endpoint that receives an RTP packet stream to be able to forward that RTP packet stream to a third party. There are some obvious security and privacy implications in supporting this, but also potential uses. This is supported in the W3C API by taking the received and decoded media and using it as media source that is re-encoding and transmitted as a new stream.

At the RTP layer, media forwarding acts as a back-to-back RTP receiver and RTP sender. The receiving side terminates the RTP session and decodes the media, while the sender side re-encodes and transmits the media using an entirely separate RTP session. The original sender will only see a single receiver of the media, and will not be able to tell that forwarding is happening based on RTP-layer information since the RTP session that is used to send the forwarded media is not connected to the RTP session on which the media was received by the node doing the forwarding.

The endpoint that is performing the forwarding is responsible for producing an RTP packet stream suitable for onwards transmission. The outgoing RTP session that is used to send the forwarded media is entirely separate to the RTP session on which the media was received. This will require media transcoding for congestion control purpose to produce a suitable bit-rate for the outgoing RTP session, reducing media quality and forcing the forwarding

endpoint to spend the resource on the transcoding. The media transcoding does result in a separation of the two different legs removing almost all dependencies, and allowing the forwarding endpoint to optimise its media transcoding operation. The cost is greatly increased computational complexity on the forwarding node. Receivers of the forwarded stream will see the forwarding device as the sender of the stream, and will not be able to tell from the RTP layer that they are receiving a forwarded stream rather than an entirely new RTP packet stream generated by the forwarding device.

#### 12.1.3. Differentiated Treatment of RTP Streams

There are use cases for differentiated treatment of RTP packet streams. Such differentiation can happen at several places in the system. First of all is the prioritization within the endpoint sending the media, which controls, both which RTP packet streams that will be sent, and their allocation of bit-rate out of the current available aggregate as determined by the congestion control.

It is expected that the WebRTC API [W3C.WD-webrtc-20130910] will allow the application to indicate relative priorities for different `MediaStreamTracks`. These priorities can then be used to influence the local RTP processing, especially when it comes to congestion control response in how to divide the available bandwidth between the RTP packet streams. Any changes in relative priority will also need to be considered for RTP packet streams that are associated with the main RTP packet streams, such as redundant streams for RTP retransmission and FEC. The importance of such redundant RTP packet streams is dependent on the media type and codec used, in regards to how robust that codec is to packet loss. However, a default policy might to be to use the same priority for redundant RTP packet stream as for the source RTP packet stream.

Secondly, the network can prioritize transport-layer flows and sub-flows, including RTP packet streams. Typically, differential treatment includes two steps, the first being identifying whether an IP packet belongs to a class that has to be treated differently, the second consisting of the actual mechanism to prioritize packets. Three common methods for classifying IP packets are:

**DiffServ:** The endpoint marks a packet with a DiffServ code point to indicate to the network that the packet belongs to a particular class.

**Flow based:** Packets that need to be given a particular treatment are identified using a combination of IP and port address.

Deep Packet Inspection: A network classifier (DPI) inspects the packet and tries to determine if the packet represents a particular application and type that is to be prioritized.

Flow-based differentiation will provide the same treatment to all packets within a transport-layer flow, i.e., relative prioritization is not possible. Moreover, if the resources are limited it might not be possible to provide differential treatment compared to best-effort for all the RTP packet streams used in a WebRTC session. The use of flow-based differentiation needs to be coordinated between the WebRTC system and the network(s). The WebRTC endpoint needs to know that flow-based differentiation might be used to provide the separation of the RTP packet streams onto different UDP flows to enable a more granular usage of flow based differentiation. The used flows, their 5-tuples and prioritization will need to be communicated to the network so that it can identify the flows correctly to enable prioritization. No specific protocol support for this is specified.

DiffServ assumes that either the endpoint or a classifier can mark the packets with an appropriate DSCP so that the packets are treated according to that marking. If the endpoint is to mark the traffic two requirements arise in the WebRTC context: 1) The WebRTC Endpoint has to know which DSCP to use and that it can use them on some set of RTP packet streams. 2) The information needs to be propagated to the operating system when transmitting the packet. Details of this process are outside the scope of this memo and are further discussed in "DSCP and other packet markings for RTCWeb QoS" [I-D.ietf-tsvwg-rtcweb-qos].

Deep Packet Inspectors will, despite the SRTP media encryption, still be fairly capable at classifying the RTP streams. The reason is that SRTP leaves the first 12 bytes of the RTP header unencrypted. This enables easy RTP stream identification using the SSRC and provides the classifier with useful information that can be correlated to determine for example the stream's media type. Using packet sizes, reception times, packet inter-spacing, RTP timestamp increments and sequence numbers, fairly reliable classifications are achieved.

For packet based marking schemes it might be possible to mark individual RTP packets differently based on the relative priority of the RTP payload. For example video codecs that have I, P, and B pictures could prioritise any payloads carrying only B frames less, as these are less damaging to lose. However, depending on the QoS mechanism and what markings that are applied, this can result in not only different packet drop probabilities but also packet reordering, see [I-D.ietf-tsvwg-rtcweb-qos] and [I-D.ietf-dart-dscp-rtp] for further discussion. As a default policy all RTP packets related to a RTP packet stream ought to be provided with the same prioritization;

per-packet prioritization is outside the scope of this memo, but might be specified elsewhere in future.

It is also important to consider how RTCP packets associated with a particular RTP packet stream need to be marked. RTCP compound packets with Sender Reports (SR), ought to be marked with the same priority as the RTP packet stream itself, so the RTCP-based round-trip time (RTT) measurements are done using the same transport-layer flow priority as the RTP packet stream experiences. RTCP compound packets containing RR packet ought to be sent with the priority used by the majority of the RTP packet streams reported on. RTCP packets containing time-critical feedback packets can use higher priority to improve the timeliness and likelihood of delivery of such feedback.

## 12.2. Media Source, RTP Streams, and Participant Identification

### 12.2.1. Media Source Identification

Each RTP packet stream is identified by a unique synchronisation source (SSRC) identifier. The SSRC identifier is carried in each of the RTP packets comprising a RTP packet stream, and is also used to identify that stream in the corresponding RTCP reports. The SSRC is chosen as discussed in Section 4.8. The first stage in demultiplexing RTP and RTCP packets received on a single transport layer flow at a WebRTC Endpoint is to separate the RTP packet streams based on their SSRC value; once that is done, additional demultiplexing steps can determine how and where to render the media.

RTP allows a mixer, or other RTP-layer middlebox, to combine encoded streams from multiple media sources to form a new encoded stream from a new media source (the mixer). The RTP packets in that new RTP packet stream can include a Contributing Source (CSRC) list, indicating which original SSRCs contributed to the combined source stream. As described in Section 4.1, implementations need to support reception of RTP data packets containing a CSRC list and RTCP packets that relate to sources present in the CSRC list. The CSRC list can change on a packet-by-packet basis, depending on the mixing operation being performed. Knowledge of what media sources contributed to a particular RTP packet can be important if the user interface indicates which participants are active in the session. Changes in the CSRC list included in packets needs to be exposed to the WebRTC application using some API, if the application is to be able to track changes in session participation. It is desirable to map CSRC values back into WebRTC MediaStream identities as they cross this API, to avoid exposing the SSRC/CSRC name space to WebRTC applications.

If the mixer-to-client audio level extension [RFC6465] is being used in the session (see Section 5.2.3), the information in the CSRC list

is augmented by audio level information for each contributing source. It is desirable to expose this information to the WebRTC application using some API, after mapping the CSRC values to WebRTC MediaStream identities, so it can be exposed in the user interface.

#### 12.2.2. SSRC Collision Detection

The RTP standard requires RTP implementations to have support for detecting and handling SSRC collisions, i.e., resolve the conflict when two different endpoints use the same SSRC value (see section 8.2 of [RFC3550]). This requirement also applies to WebRTC Endpoints. There are several scenarios where SSRC collisions can occur:

- o In a point-to-point session where each SSRC is associated with either of the two endpoints and where the main media carrying SSRC identifier will be announced in the signalling channel, a collision is less likely to occur due to the information about used SSRCs. If SDP is used, this information is provided by Source-Specific SDP Attributes [RFC5576]. Still, collisions can occur if both endpoints start using a new SSRC identifier prior to having signalled it to the peer and received acknowledgement on the signalling message. The Source-Specific SDP Attributes [RFC5576] contains a mechanism to signal how the endpoint resolved the SSRC collision.
- o SSRC values that have not been signalled could also appear in an RTP session. This is more likely than it appears, since some RTP functions use extra SSRCs to provide their functionality. For example, retransmission data might be transmitted using a separate RTP packet stream that requires its own SSRC, separate to the SSRC of the source RTP packet stream [RFC4588]. In those cases, an endpoint can create a new SSRC that strictly doesn't need to be announced over the signalling channel to function correctly on both RTP and RTCPeerConnection level.
- o Multiple endpoints in a multiparty conference can create new sources and signal those towards the RTP middlebox. In cases where the SSRC/CSRC are propagated between the different endpoints from the RTP middlebox collisions can occur.
- o An RTP middlebox could connect an endpoint's RTCPeerConnection to another RTCPeerConnection from the same endpoint, thus forming a loop where the endpoint will receive its own traffic. While it is clearly considered a bug, it is important that the endpoint is able to recognise and handle the case when it occurs. This case becomes even more problematic when media mixers, and so on, are involved, where the stream received is a different stream but still contains this client's input.

These SSRC/CSRC collisions can only be handled on RTP level as long as the same RTP session is extended across multiple RTCPeerConnections by a RTP middlebox. To resolve the more generic case where multiple RTCPeerConnections are interconnected, identification of the media source(s) part of a MediaStreamTrack being propagated across multiple interconnected RTCPeerConnection needs to be preserved across these interconnections.

#### 12.2.3. Media Synchronisation Context

When an endpoint sends media from more than one media source, it needs to consider if (and which of) these media sources are to be synchronized. In RTP/RTCP, synchronisation is provided by having a set of RTP packet streams be indicated as coming from the same synchronisation context and logical endpoint by using the same RTCP CNAME identifier.

The next provision is that the internal clocks of all media sources, i.e., what drives the RTP timestamp, can be correlated to a system clock that is provided in RTCP Sender Reports encoded in an NTP format. By correlating all RTP timestamps to a common system clock for all sources, the timing relation of the different RTP packet streams, also across multiple RTP sessions can be derived at the receiver and, if desired, the streams can be synchronized. The requirement is for the media sender to provide the correlation information; it is up to the receiver to use it or not.

### 13. Security Considerations

The overall security architecture for WebRTC is described in [I-D.ietf-rtcweb-security-arch], and security considerations for the WebRTC framework are described in [I-D.ietf-rtcweb-security]. These considerations also apply to this memo.

The security considerations of the RTP specification, the RTP/SAVPF profile, and the various RTP/RTCP extensions and RTP payload formats that form the complete protocol suite described in this memo apply. It is not believed there are any new security considerations resulting from the combination of these various protocol extensions.

The Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback [RFC5124] (RTP/SAVPF) provides handling of fundamental issues by offering confidentiality, integrity and partial source authentication. A mandatory to implement and use media security solution is created by combining this secured RTP profile and DTLS-SRTP keying [RFC5764] as defined by Section 5.5 of [I-D.ietf-rtcweb-security-arch].

RTCP packets convey a Canonical Name (CNAME) identifier that is used to associate RTP packet streams that need to be synchronised across related RTP sessions. Inappropriate choice of CNAME values can be a privacy concern, since long-term persistent CNAME identifiers can be used to track users across multiple WebRTC calls. Section 4.9 of this memo mandates generation of short-term persistent RTCP CNAMEs, as specified in RFC7022, resulting in untraceable CNAME values that alleviate this risk.

Some potential denial of service attacks exist if the RTCP reporting interval is configured to an inappropriate value. This could be done by configuring the RTCP bandwidth fraction to an excessively large or small value using the SDP "b=RR:" or "b=RS:" lines [RFC3556], or some similar mechanism, or by choosing an excessively large or small value for the RTP/AVPF minimal receiver report interval (if using SDP, this is the "a=rtcp-fb:... trr-int" parameter) [RFC4585]. The risks are as follows:

1. the RTCP bandwidth could be configured to make the regular reporting interval so large that effective congestion control cannot be maintained, potentially leading to denial of service due to congestion caused by the media traffic;
2. the RTCP interval could be configured to a very small value, causing endpoints to generate high rate RTCP traffic, potentially leading to denial of service due to the non-congestion controlled RTCP traffic; and
3. RTCP parameters could be configured differently for each endpoint, with some of the endpoints using a large reporting interval and some using a smaller interval, leading to denial of service due to premature participant timeouts due to mismatched timeout periods which are based on the reporting interval (this is a particular concern if endpoints use a small but non-zero value for the RTP/AVPF minimal receiver report interval (trr-int) [RFC4585], as discussed in Section 6.1 of [I-D.ietf-avtcore-rtp-multi-stream]).

Premature participant timeout can be avoided by using the fixed (non-reduced) minimum interval when calculating the participant timeout (see Section 4.1 of this memo and Section 6.1 of [I-D.ietf-avtcore-rtp-multi-stream]). To address the other concerns, endpoints SHOULD ignore parameters that configure the RTCP reporting interval to be significantly longer than the default five second interval specified in [RFC3550] (unless the media data rate is so low that the longer reporting interval roughly corresponds to 5% of the media data rate), or that configure the RTCP reporting interval small enough that the RTCP bandwidth would exceed the media bandwidth.

The guidelines in [RFC6562] apply when using variable bit rate (VBR) audio codecs such as Opus (see Section 4.3 for discussion of mandated audio codecs). The guidelines in [RFC6562] also apply, but are of lesser importance, when using the client-to-mixer audio level header extensions (Section 5.2.2) or the mixer-to-client audio level header extensions (Section 5.2.3). The use of the encryption of the header extensions are RECOMMENDED, unless there are known reasons, like RTP middleboxes performing voice activity based source selection or third party monitoring that will greatly benefit from the information, and this has been expressed using API or signalling. If further evidence are produced to show that information leakage is significant from audio level indications, then use of encryption needs to be mandated at that time.

In multi-party communication scenarios using RTP Middleboxes, a lot of trust is placed on these middleboxes to preserve the sessions security. The middlebox needs to maintain the confidentiality, integrity and perform source authentication. As discussed in Section 12.1.1 the middlebox can perform checks that prevents any endpoint participating in a conference to impersonate another. Some additional security considerations regarding multi-party topologies can be found in [I-D.ietf-avtcore-rtp-topologies-update].

#### 14. IANA Considerations

This memo makes no request of IANA.

Note to RFC Editor: this section is to be removed on publication as an RFC.

#### 15. Acknowledgements

The authors would like to thank Bernard Aboba, Harald Alvestrand, Cary Bran, Ben Campbell, Alissa Cooper, Spencer Dawkins, Charles Eckel, Alex Eleftheriadis, Christian Groves, Chris Inacio, Cullen Jennings, Olle Johansson, Suhas Nandakumar, Dan Romascanu, Jim Spring, Martin Thomson, and the other members of the IETF RTCWEB working group for their valuable feedback.

#### 16. References

##### 16.1. Normative References

[I-D.ietf-avtcore-multi-media-rtp-session]  
Westerlund, M., Perkins, C., and J. Lennox, "Sending Multiple Types of Media in a Single RTP Session", draft-ietf-avtcore-multi-media-rtp-session-13 (work in progress), December 2015.

- [I-D.ietf-avtc core-rtp-circuit-breakers]  
Perkins, C. and V. Varun, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", draft-ietf-avtc core-rtp-circuit-breakers-13 (work in progress), February 2016.
- [I-D.ietf-avtc core-rtp-multi-stream]  
Lennox, J., Westerlund, M., Wu, Q., and C. Perkins, "Sending Multiple RTP Streams in a Single RTP Session", draft-ietf-avtc core-rtp-multi-stream-11 (work in progress), December 2015.
- [I-D.ietf-avtc core-rtp-multi-stream-optimisation]  
Lennox, J., Westerlund, M., Wu, Q., and C. Perkins, "Sending Multiple RTP Streams in a Single RTP Session: Grouping RTCP Reception Statistics and Other Feedback", draft-ietf-avtc core-rtp-multi-stream-optimisation-12 (work in progress), March 2016.
- [I-D.ietf-avtc core-rtp-topologies-update]  
Westerlund, M. and S. Wenger, "RTP Topologies", draft-ietf-avtc core-rtp-topologies-update-10 (work in progress), July 2015.
- [I-D.ietf-mmusic-mux-exclusive]  
Holmberg, C., "Indicating Exclusive Support of RTP/RTCP Multiplexing using SDP", draft-ietf-mmusic-mux-exclusive-03 (work in progress), February 2016.
- [I-D.ietf-mmusic-sdp-bundle-negotiation]  
Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-negotiation-27 (work in progress), February 2016.
- [I-D.ietf-rtcweb-audio]  
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-10 (work in progress), February 2016.
- [I-D.ietf-rtcweb-fec]  
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-02 (work in progress), October 2015.

- [I-D.ietf-rtcweb-overview]  
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-15 (work in progress), January 2016.
- [I-D.ietf-rtcweb-security]  
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-08 (work in progress), February 2015.
- [I-D.ietf-rtcweb-security-arch]  
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-11 (work in progress), March 2015.
- [I-D.ietf-rtcweb-video]  
Roach, A., "WebRTC Video Processing and Codec Requirements", draft-ietf-rtcweb-video-06 (work in progress), June 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2736] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, DOI 10.17487/RFC2736, December 1999, <<http://www.rfc-editor.org/info/rfc2736>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<http://www.rfc-editor.org/info/rfc3550>>.
- [RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, DOI 10.17487/RFC3551, July 2003, <<http://www.rfc-editor.org/info/rfc3551>>.
- [RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, DOI 10.17487/RFC3556, July 2003, <<http://www.rfc-editor.org/info/rfc3556>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<http://www.rfc-editor.org/info/rfc3711>>.

- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<http://www.rfc-editor.org/info/rfc4566>>.
- [RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <<http://www.rfc-editor.org/info/rfc4585>>.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, DOI 10.17487/RFC4588, July 2006, <<http://www.rfc-editor.org/info/rfc4588>>.
- [RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, DOI 10.17487/RFC4961, July 2007, <<http://www.rfc-editor.org/info/rfc4961>>.
- [RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, DOI 10.17487/RFC5104, February 2008, <<http://www.rfc-editor.org/info/rfc5104>>.
- [RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, DOI 10.17487/RFC5124, February 2008, <<http://www.rfc-editor.org/info/rfc5124>>.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, DOI 10.17487/RFC5285, July 2008, <<http://www.rfc-editor.org/info/rfc5285>>.
- [RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, DOI 10.17487/RFC5506, April 2009, <<http://www.rfc-editor.org/info/rfc5506>>.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <<http://www.rfc-editor.org/info/rfc5761>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<http://www.rfc-editor.org/info/rfc5764>>.

- [RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, DOI 10.17487/RFC6051, November 2010, <<http://www.rfc-editor.org/info/rfc6051>>.
- [RFC6464] Lennox, J., Ed., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, DOI 10.17487/RFC6464, December 2011, <<http://www.rfc-editor.org/info/rfc6464>>.
- [RFC6465] Ivov, E., Ed., Marocco, E., Ed., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, DOI 10.17487/RFC6465, December 2011, <<http://www.rfc-editor.org/info/rfc6465>>.
- [RFC6562] Perkins, C. and JM. Valin, "Guidelines for the Use of Variable Bit Rate Audio with Secure RTP", RFC 6562, DOI 10.17487/RFC6562, March 2012, <<http://www.rfc-editor.org/info/rfc6562>>.
- [RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, DOI 10.17487/RFC6904, April 2013, <<http://www.rfc-editor.org/info/rfc6904>>.
- [RFC7007] Terriberry, T., "Update to Remove DVI4 from the Recommended Codecs for the RTP Profile for Audio and Video Conferences with Minimal Control (RTP/AVP)", RFC 7007, DOI 10.17487/RFC7007, August 2013, <<http://www.rfc-editor.org/info/rfc7007>>.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, DOI 10.17487/RFC7022, September 2013, <<http://www.rfc-editor.org/info/rfc7022>>.
- [RFC7160] Petit-Huguenin, M. and G. Zorn, Ed., "Support for Multiple Clock Rates in an RTP Session", RFC 7160, DOI 10.17487/RFC7160, April 2014, <<http://www.rfc-editor.org/info/rfc7160>>.
- [RFC7164] Gross, K. and R. Brandenburg, "RTP and Leap Seconds", RFC 7164, DOI 10.17487/RFC7164, March 2014, <<http://www.rfc-editor.org/info/rfc7164>>.

[W3C.WD-mediacapture-streams-20130903]

Burnett, D., Bergkvist, A., Jennings, C., and A. Narayanan, "Media Capture and Streams", World Wide Web Consortium WD WD-mediacapture-streams-20130903, September 2013, <<http://www.w3.org/TR/2013/WD-mediacapture-streams-20130903>>.

[W3C.WD-webrtc-20130910]

Bergkvist, A., Burnett, D., Jennings, C., and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20130910, September 2013, <<http://www.w3.org/TR/2013/WD-webrtc-20130910>>.

## 16.2. Informative References

[I-D.ietf-avtcore-multiplex-guidelines]

Westerlund, M., Perkins, C., and H. Alvestrand, "Guidelines for using the Multiplexing Features of RTP to Support Multiple Media Streams", draft-ietf-avtcore-multiplex-guidelines-03 (work in progress), October 2014.

[I-D.ietf-avtext-rtp-grouping-taxonomy]

Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G., and B. Burman, "A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources", draft-ietf-avtext-rtp-grouping-taxonomy-08 (work in progress), July 2015.

[I-D.ietf-dart-dscp-rtp]

Black, D. and P. Jones, "Differentiated Services (DiffServ) and Real-time Communication", draft-ietf-dart-dscp-rtp-10 (work in progress), November 2014.

[I-D.ietf-mmusic-msid]

Alvestrand, H., "WebRTC MediaStream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-11 (work in progress), October 2015.

[I-D.ietf-payload-rtp-howto]

Westerlund, M., "How to Write an RTP Payload Format", draft-ietf-payload-rtp-howto-14 (work in progress), May 2015.

[I-D.ietf-rmcat-cc-requirements]

Jesup, R. and Z. Sarker, "Congestion Control Requirements for Interactive Real-Time Media", draft-ietf-rmcat-cc-requirements-09 (work in progress), December 2014.

- [I-D.ietf-rtcweb-jsep]  
Uberti, J., Jennings, C., and E. Rescorla, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-13 (work in progress), March 2016.
- [I-D.ietf-tsvwg-rtcweb-qos]  
Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "DSCP and other packet markings for WebRTC QoS", draft-ietf-tsvwg-rtcweb-qos-14 (work in progress), March 2016.
- [RFC3611] Friedman, T., Ed., Caceres, R., Ed., and A. Clark, Ed., "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, DOI 10.17487/RFC3611, November 2003, <<http://www.rfc-editor.org/info/rfc3611>>.
- [RFC4383] Baugher, M. and E. Carrara, "The Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Secure Real-time Transport Protocol (SRTP)", RFC 4383, DOI 10.17487/RFC4383, February 2006, <<http://www.rfc-editor.org/info/rfc4383>>.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.
- [RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, DOI 10.17487/RFC5576, June 2009, <<http://www.rfc-editor.org/info/rfc5576>>.
- [RFC5968] Ott, J. and C. Perkins, "Guidelines for Extending the RTP Control Protocol (RTCP)", RFC 5968, DOI 10.17487/RFC5968, September 2010, <<http://www.rfc-editor.org/info/rfc5968>>.
- [RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, DOI 10.17487/RFC6263, June 2011, <<http://www.rfc-editor.org/info/rfc6263>>.
- [RFC6792] Wu, Q., Ed., Hunt, G., and P. Arden, "Guidelines for Use of the RTP Monitoring Framework", RFC 6792, DOI 10.17487/RFC6792, November 2012, <<http://www.rfc-editor.org/info/rfc6792>>.

[RFC7478] Holmberg, C., Hakanesson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <<http://www.rfc-editor.org/info/rfc7478>>.

#### Authors' Addresses

Colin Perkins  
University of Glasgow  
School of Computing Science  
Glasgow G12 8QQ  
United Kingdom

Email: [csp@csp Perkins.org](mailto:csp@csp Perkins.org)  
URI: <https://csp Perkins.org/>

Magnus Westerlund  
Ericsson  
Farogatan 6  
SE-164 80 Kista  
Sweden

Phone: +46 10 714 82 87  
Email: [magnus.westerlund@ericsson.com](mailto:magnus.westerlund@ericsson.com)

Joerg Ott  
Aalto University  
School of Electrical Engineering  
Espoo 02150  
Finland

Email: [jorg.ott@aalto.fi](mailto:jorg.ott@aalto.fi)

RTCWEB  
Internet-Draft  
Intended status: Standards Track  
Expires: January 22, 2020

E. Rescorla  
RTFM, Inc.  
July 21, 2019

WebRTC Security Architecture  
draft-ietf-rtcweb-security-arch-20

Abstract

This document defines the security architecture for WebRTC, a protocol suite intended for use with real-time applications that can be deployed in browsers - "real time communication on the Web".

Status of This Memo

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

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

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

This Internet-Draft will expire on January 22, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	5
3. Trust Model . . . . .	5
3.1. Authenticated Entities . . . . .	5
3.2. Unauthenticated Entities . . . . .	6
4. Overview . . . . .	6
4.1. Initial Signaling . . . . .	8
4.2. Media Consent Verification . . . . .	10
4.3. DTLS Handshake . . . . .	11
4.4. Communications and Consent Freshness . . . . .	11
5. SDP Identity Attribute . . . . .	12
5.1. Offer/Answer Considerations . . . . .	13
5.1.1. Generating the Initial SDP Offer . . . . .	13
5.1.2. Generating of SDP Answer . . . . .	14
5.1.3. Processing an SDP Offer or Answer . . . . .	14
5.1.4. Modifying the Session . . . . .	14
6. Detailed Technical Description . . . . .	14
6.1. Origin and Web Security Issues . . . . .	14
6.2. Device Permissions Model . . . . .	15
6.3. Communications Consent . . . . .	17
6.4. IP Location Privacy . . . . .	17
6.5. Communications Security . . . . .	18
7. Web-Based Peer Authentication . . . . .	20
7.1. Trust Relationships: IdPs, APs, and RPs . . . . .	21
7.2. Overview of Operation . . . . .	23
7.3. Items for Standardization . . . . .	24
7.4. Binding Identity Assertions to JSEP Offer/Answer Transactions . . . . .	24
7.4.1. Carrying Identity Assertions . . . . .	25
7.5. Determining the IdP URI . . . . .	26
7.5.1. Authenticating Party . . . . .	27
7.5.2. Relying Party . . . . .	28
7.6. Requesting Assertions . . . . .	28
7.7. Managing User Login . . . . .	29

8. Verifying Assertions . . . . .	29
8.1. Identity Formats . . . . .	30
9. Security Considerations . . . . .	31
9.1. Communications Security . . . . .	31
9.2. Privacy . . . . .	32
9.3. Denial of Service . . . . .	33
9.4. IdP Authentication Mechanism . . . . .	34
9.4.1. PeerConnection Origin Check . . . . .	34
9.4.2. IdP Well-known URI . . . . .	34
9.4.3. Privacy of IdP-generated identities and the hosting site . . . . .	35
9.4.4. Security of Third-Party IdPs . . . . .	35
9.4.4.1. Confusable Characters . . . . .	35
9.4.5. Web Security Feature Interactions . . . . .	35
9.4.5.1. Popup Blocking . . . . .	36
9.4.5.2. Third Party Cookies . . . . .	36
10. IANA Considerations . . . . .	36
11. Acknowledgements . . . . .	37
12. Changes . . . . .	37
12.1. Changes since -15 . . . . .	37
12.2. Changes since -11 . . . . .	37
12.3. Changes since -10 . . . . .	37
12.4. Changes since -06 . . . . .	37
12.5. Changes since -05 . . . . .	38
12.6. Changes since -03 . . . . .	38
12.7. Changes since -03 . . . . .	38
12.8. Changes since -02 . . . . .	38
13. References . . . . .	39
13.1. Normative References . . . . .	39
13.2. Informative References . . . . .	42
Author's Address . . . . .	43

## 1. Introduction

The Real-Time Communications on the Web (RTCWEB) working group standardized protocols for real-time communications between Web browsers, generally called "WebRTC" [I-D.ietf-rtcweb-overview]. The major use cases for WebRTC technology are real-time audio and/or video calls, Web conferencing, and direct data transfer. Unlike most conventional real-time systems, (e.g., SIP-based [RFC3261] soft phones) WebRTC communications are directly controlled by some Web server, via a JavaScript (JS) API as shown in Figure 1.

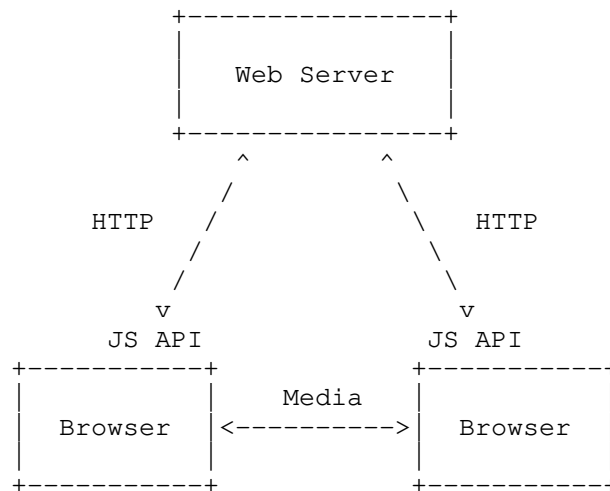


Figure 1: A simple WebRTC system

A more complicated system might allow for interdomain calling, as shown in Figure 2. The protocol to be used between the domains is not standardized by WebRTC, but given the installed base and the form of the WebRTC API is likely to be something SDP-based like SIP or something like Extensible Messaging and Presence Protocol (XMPP) [RFC6120].

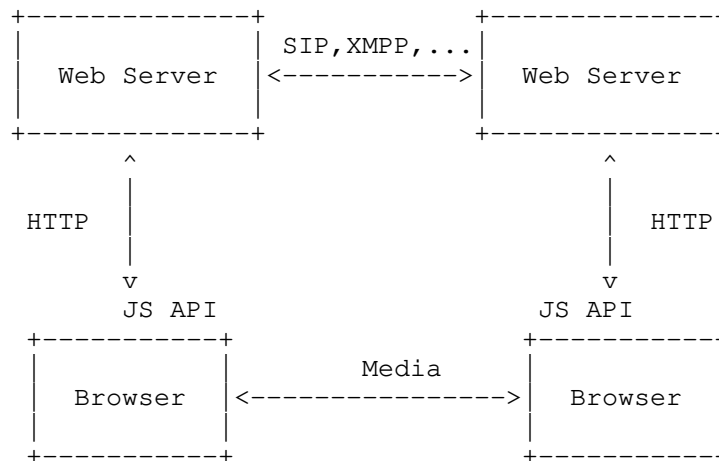


Figure 2: A multidomain WebRTC system

This system presents a number of new security challenges, which are analyzed in [I-D.ietf-rtcweb-security]. This document describes a

security architecture for WebRTC which addresses the threats and requirements described in that document.

## 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. Trust Model

The basic assumption of this architecture is that network resources exist in a hierarchy of trust, rooted in the browser, which serves as the user's Trusted Computing Base (TCB). Any security property which the user wishes to have enforced must be ultimately guaranteed by the browser (or transitively by some property the browser verifies). Conversely, if the browser is compromised, then no security guarantees are possible. Note that there are cases (e.g., Internet kiosks) where the user can't really trust the browser that much. In these cases, the level of security provided is limited by how much they trust the browser.

Optimally, we would not rely on trust in any entities other than the browser. However, this is unfortunately not possible if we wish to have a functional system. Other network elements fall into two categories: those which can be authenticated by the browser and thus can be granted permissions to access sensitive resources, and those which cannot be authenticated and thus are untrusted.

### 3.1. Authenticated Entities

There are two major classes of authenticated entities in the system:

- o Calling services: Web sites whose origin we can verify (optimally via HTTPS, but in some cases because we are on a topologically restricted network, such as behind a firewall, and can infer authentication from firewall behavior).
- o Other users: WebRTC peers whose origin we can verify cryptographically (optimally via DTLS-SRTP).

Note that merely being authenticated does not make these entities trusted. For instance, just because we can verify that <https://www.example.org/> is owned by Dr. Evil does not mean that we can trust Dr. Evil to access our camera and microphone. However, it gives the user an opportunity to determine whether he wishes to trust

Dr. Evil or not; after all, if he desires to contact Dr. Evil (perhaps to arrange for ransom payment), it's safe to temporarily give him access to the camera and microphone for the purpose of the call, but he doesn't want Dr. Evil to be able to access his camera and microphone other than during the call. The point here is that we must first identify other elements before we can determine whether and how much to trust them. Additionally, sometimes we need to identify the communicating peer before we know what policies to apply.

### 3.2. Unauthenticated Entities

Other than the above entities, we are not generally able to identify other network elements, thus we cannot trust them. This does not mean that it is not possible to have any interaction with them, but it means that we must assume that they will behave maliciously and design a system which is secure even if they do so.

## 4. Overview

This section describes a typical WebRTC session and shows how the various security elements interact and what guarantees are provided to the user. The example in this section is a "best case" scenario in which we provide the maximal amount of user authentication and media privacy with the minimal level of trust in the calling service. Simpler versions with lower levels of security are also possible and are noted in the text where applicable. It's also important to recognize the tension between security (or performance) and privacy. The example shown here is aimed towards settings where we are more concerned about secure calling than about privacy, but as we shall see, there are settings where one might wish to make different tradeoffs--this architecture is still compatible with those settings.

For the purposes of this example, we assume the topology shown in the figures below. This topology is derived from the topology shown in Figure 1, but separates Alice and Bob's identities from the process of signaling. Specifically, Alice and Bob have relationships with some Identity Provider (IdP) that supports a protocol (such as OpenID Connect) that can be used to demonstrate their identity to other parties. For instance, Alice might have an account with a social network which she can then use to authenticate to other web sites without explicitly having an account with those sites; this is a fairly conventional pattern on the Web. Section 7.1 provides an overview of Identity Providers and the relevant terminology. Alice and Bob might have relationships with different IdPs as well.

This separation of identity provision and signaling isn't particularly important in "closed world" cases where Alice and Bob

are users on the same social network and have identities based on that domain (Figure 3). However, there are important settings where that is not the case, such as federation (calls from one domain to another; Figure 4) and calling on untrusted sites, such as where two users who have a relationship via a given social network want to call each other on another, untrusted, site, such as a poker site.

Note that the servers themselves are also authenticated by an external identity service, the SSL/TLS certificate infrastructure (not shown). As is conventional in the Web, all identities are ultimately rooted in that system. For instance, when an IdP makes an identity assertion, the Relying Party consuming that assertion is able to verify because it is able to connect to the IdP via HTTPS.

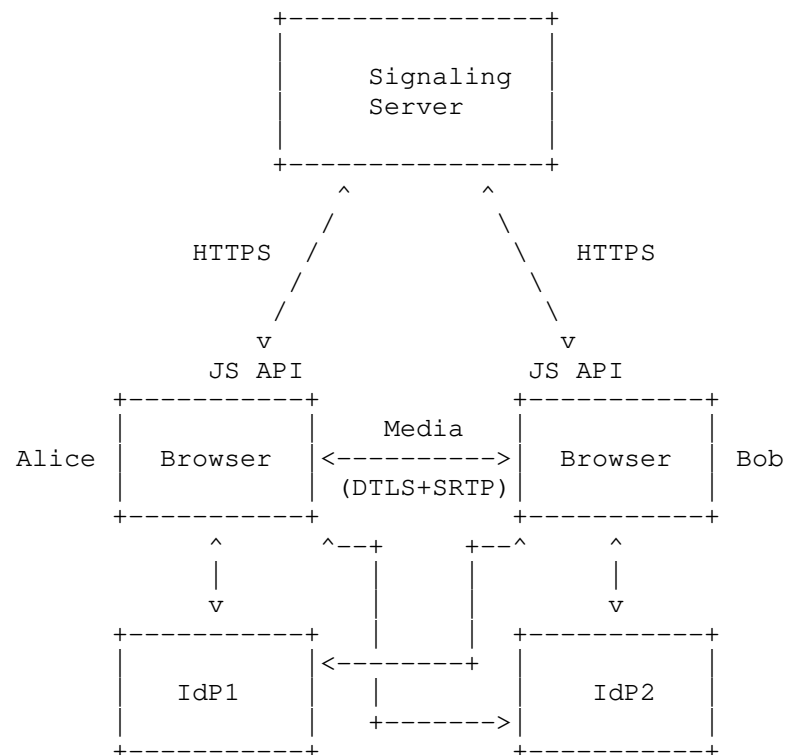


Figure 3: A call with IdP-based identity

Figure 4 shows essentially the same calling scenario but with a call between two separate domains (i.e., a federated case), as in Figure 2. As mentioned above, the domains communicate by some unspecified protocol and providing separate signaling and identity

allows for calls to be authenticated regardless of the details of the inter-domain protocol.

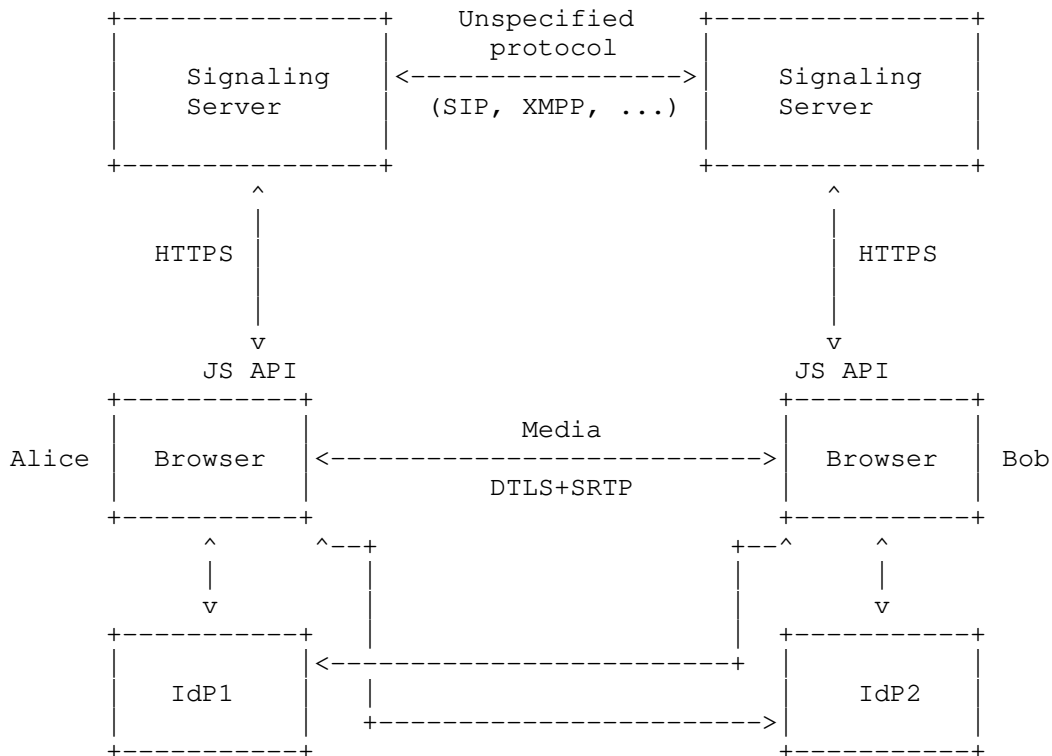


Figure 4: A federated call with IdP-based identity

#### 4.1. Initial Signaling

For simplicity, assume the topology in Figure 3. Alice and Bob are both users of a common calling service; they both have approved the calling service to make calls (we defer the discussion of device access permissions until later). They are both connected to the calling service via HTTPS and so know the origin with some level of confidence. They also have accounts with some identity provider. This sort of identity service is becoming increasingly common in the Web environment (with technologies such as Federated Google Login, Facebook Connect, OAuth, OpenID, WebFinger), and is often provided as a side effect service of a user's ordinary accounts with some service. In this example, we show Alice and Bob using a separate identity service, though the identity service may be the same entity as the calling service or there may be no identity service at all.

Alice is logged onto the calling service and decides to call Bob. She can see from the calling service that he is online and the calling service presents a JS UI in the form of a button next to Bob's name which says "Call". Alice clicks the button, which initiates a JS callback that instantiates a `PeerConnection` object. This does not require a security check: JS from any origin is allowed to get this far.

Once the `PeerConnection` is created, the calling service JS needs to set up some media. Because this is an audio/video call, it creates a `MediaStream` with two `MediaStreamTracks`, one connected to an audio input and one connected to a video input. At this point the first security check is required: untrusted origins are not allowed to access the camera and microphone, so the browser prompts Alice for permission.

In the current W3C API, once some streams have been added, Alice's browser + JS generates a signaling message [I-D.ietf-rtcweb-jsep] containing:

- o Media channel information
- o Interactive Connectivity Establishment (ICE) [RFC8445] candidates
- o A fingerprint attribute binding the communication to a key pair [RFC5763]. Note that this key may simply be ephemerally generated for this call or specific to this domain, and Alice may have a large number of such keys.

Prior to sending out the signaling message, the `PeerConnection` code contacts the identity service and obtains an assertion binding Alice's identity to her fingerprint. The exact details depend on the identity service (though as discussed in Section 7 `PeerConnection` can be agnostic to them), but for now it's easiest to think of as an OAuth token. The assertion may bind other information to the identity besides the fingerprint, but at minimum it needs to bind the fingerprint.

This message is sent to the signaling server, e.g., by `XMLHttpRequest` [`XmlHttpRequest`] or by `WebSockets` [RFC6455], over TLS [RFC5246]. The signaling server processes the message from Alice's browser, determines that this is a call to Bob and sends a signaling message to Bob's browser (again, the format is currently undefined). The JS on Bob's browser processes it, and alerts Bob to the incoming call and to Alice's identity. In this case, Alice has provided an identity assertion and so Bob's browser contacts Alice's identity provider (again, this is done in a generic way so the browser has no specific knowledge of the IdP) to verify the assertion. It is also

possible to have IdPs with which the browser has a specific trustrelationship, as described in Section 7.1. This allows the browser to display a trusted element in the browser chrome indicating that a call is coming in from Alice. If Alice is in Bob's address book, then this interface might also include her real name, a picture, etc. The calling site will also provide some user interface element (e.g., a button) to allow Bob to answer the call, though this is most likely not part of the trusted UI.

If Bob agrees a PeerConnection is instantiated with the message from Alice's side. Then, a similar process occurs as on Alice's browser: Bob's browser prompts him for device permission, the media streams are created, and a return signaling message containing media information, ICE candidates, and a fingerprint is sent back to Alice via the signaling service. If Bob has a relationship with an IdP, the message will also come with an identity assertion.

At this point, Alice and Bob each know that the other party wants to have a secure call with them. Based purely on the interface provided by the signaling server, they know that the signaling server claims that the call is from Alice to Bob. This level of security is provided merely by having the fingerprint in the message and having that message received securely from the signaling server. Because the far end sent an identity assertion along with their message, they know that this is verifiable from the IdP as well. Note that if the call is federated, as shown in Figure 4 then Alice is able to verify Bob's identity in a way that is not mediated by either her signaling server or Bob's. Rather, she verifies it directly with Bob's IdP.

Of course, the call works perfectly well if either Alice or Bob doesn't have a relationship with an IdP; they just get a lower level of assurance. I.e., they simply have whatever information their calling site claims about the caller/callee's identity. Moreover, Alice might wish to make an anonymous call through an anonymous calling site, in which case she would of course just not provide any identity assertion and the calling site would mask her identity from Bob.

#### 4.2. Media Consent Verification

As described in ([I-D.ietf-rtcweb-security]; Section 4.2) media consent verification is provided via ICE. Thus, Alice and Bob perform ICE checks with each other. At the completion of these checks, they are ready to send non-ICE data.

At this point, Alice knows that (a) Bob (assuming he is verified via his IdP) or someone else who the signaling service is claiming is Bob is willing to exchange traffic with her and (b) that either Bob is at

the IP address which she has verified via ICE or there is an attacker who is on-path to that IP address detouring the traffic. Note that it is not possible for an attacker who is on-path between Alice and Bob but not attached to the signaling service to spoof these checks because they do not have the ICE credentials. Bob has the same security guarantees with respect to Alice.

#### 4.3. DTLS Handshake

Once the requisite ICE checks have completed, Alice and Bob can set up a secure channel or channels. This is performed via DTLS [RFC6347] and DTLS-SRTP [RFC5763] keying for SRTP [RFC3711] for the media channel and SCTP over DTLS [RFC8261] for data channels. Specifically, Alice and Bob perform a DTLS handshake on every component which has been established by ICE. The total number of channels depends on the amount of muxing; in the most likely case we are using both RTP/RTCP mux and muxing multiple media streams on the same channel, in which case there is only one DTLS handshake. Once the DTLS handshake has completed, the keys are exported [RFC5705] and used to key SRTP for the media channels.

At this point, Alice and Bob know that they share a set of secure data and/or media channels with keys which are not known to any third-party attacker. If Alice and Bob authenticated via their IdPs, then they also know that the signaling service is not mounting a man-in-the-middle attack on their traffic. Even if they do not use an IdP, as long as they have minimal trust in the signaling service not to perform a man-in-the-middle attack, they know that their communications are secure against the signaling service as well (i.e., that the signaling service cannot mount a passive attack on the communications).

#### 4.4. Communications and Consent Freshness

From a security perspective, everything from here on in is a little anticlimactic: Alice and Bob exchange data protected by the keys negotiated by DTLS. Because of the security guarantees discussed in the previous sections, they know that the communications are encrypted and authenticated.

The one remaining security property we need to establish is "consent freshness", i.e., allowing Alice to verify that Bob is still prepared to receive her communications so that Alice does not continue to send large traffic volumes to entities which went abruptly offline. ICE specifies periodic STUN keepalives but only if media is not flowing. Because the consent issue is more difficult here, we require WebRTC implementations to periodically send keepalives. As described in Section 5.3, these keepalives MUST be based on the consent freshness

mechanism specified in [RFC7675]. If a keepalive fails and no new ICE channels can be established, then the session is terminated.

## 5. SDP Identity Attribute

The SDP 'identity' attribute is a session-level attribute that is used by an endpoint to convey its identity assertion to its peer. The identity assertion value is encoded as Base-64, as described in Section 4 of [RFC4648].

The procedures in this section are based on the assumption that the identity assertion of an endpoint is bound to the fingerprints of the endpoint. This does not preclude the definition of alternative means of binding an assertion to the endpoint, but such means are outside the scope of this specification.

The semantics of multiple 'identity' attributes within an offer or answer are undefined. Implementations SHOULD only include a single 'identity' attribute in an offer or answer and relying parties MAY elect to ignore all but the first 'identity' attribute.

Name: identity

Value: identity-assertion

Usage Level: session

Charset Dependent: no

Default Value: N/A

Name: identity

## Syntax:

```

identity-assertion      = identity-assertion-value
                           *(SP identity-extension)
identity-assertion-value = base64
identity-extension      = extension-name [ "=" extension-value ]
extension-name          = token
extension-value         = 1*(%x01-09 / %x0b-0c / %x0e-3a / %x3c-ff)
                           ; byte-string from [RFC4566]

```

<ALPHA and DIGIT as defined in [RFC4566]>

<base64 as defined in [RFC4566]>

## Example:

```

a=identity:\
  eyJpZHAiOnsiZG9tYWluIjoizXhhbXBsZS5vcmdiLCJwcm90b2NvbCI6ImJvZ3Vz\
  In0sImFzc2VydGlvbii6IntcImlkZW50aXR5XCI6XCJib2JAZXhhbXBsZS5vcmdc\
  IixcImNvbnRlbnRzXCI6XCJhYmNkZWZnaGlqa2xtbm9wcXJzdHV2d3l6XCIsXCJz\
  aWduYXRlcmVcIjpcIjAxMDIwMzA0MDUwNlwiSj9

```

Note that long lines in the example are folded to meet the column width constraints of this document; the backslash ("\") at the end of a line, the carriage return that follows, and whitespace shall be ignored.

This specification does not define any extensions for the attribute.

The identity-assertion value is a JSON [RFC8259] encoded string. The JSON object contains two keys: "assertion" and "idp". The "assertion" key value contains an opaque string that is consumed by the IdP. The "idp" key value contains a dictionary with one or two further values that identify the IdP. See Section 7.6 for more details.

## 5.1. Offer/Answer Considerations

This section defines the SDP Offer/Answer [RFC3264] considerations for the SDP 'identity' attribute.

Within this section, 'initial offer' refers to the first offer in the SDP session that contains an SDP "identity" attribute.

## 5.1.1. Generating the Initial SDP Offer

When an offerer sends an offer, in order to provide its identity assertion to the peer, it includes an 'identity' attribute in the offer. In addition, the offerer includes one or more SDP

'fingerprint' attributes. The 'identity' attribute MUST be bound to all the 'fingerprint' attributes in the session description.

#### 5.1.2. Generating of SDP Answer

If the answerer elects to include an 'identity' attribute, it follows the same steps as those in Section 5.1.1. The answerer can choose to include or omit an 'identity' attribute independently, regardless of whether the offerer did so.

#### 5.1.3. Processing an SDP Offer or Answer

When an endpoint receives an offer or answer that contains an 'identity' attribute, the answerer can use the the attribute information to contact the IdP and verify the identity of the peer. If the identity requires a third-party IdP as described in Section 7.1 then that IdP will need to have been specifically configured. If the identity verification fails, the answerer MUST discard the offer or answer as malformed.

#### 5.1.4. Modifying the Session

When modifying a session, if the set of fingerprints is unchanged, then the sender MAY send the same 'identity' attribute. In this case, the established identity MUST be applied to existing DTLS connections as well as new connections established using one of those fingerprints. Note that [I-D.ietf-rtcweb-jsep], Section 5.2.1 requires that each media section use the same set of fingerprints for every media section. If a new identity attribute is received, then the receiver MUST apply that identity to all existing connections.

If the set of fingerprints changes, then the sender MUST either send a new 'identity' attribute or none at all. Because a change in fingerprints also causes a new DTLS connection to be established, the receiver MUST discard all previously established identities.

### 6. Detailed Technical Description

#### 6.1. Origin and Web Security Issues

The basic unit of permissions for WebRTC is the origin [RFC6454]. Because the security of the origin depends on being able to authenticate content from that origin, the origin can only be securely established if data is transferred over HTTPS [RFC2818]. Thus, clients MUST treat HTTP and HTTPS origins as different permissions domains. Note: this follows directly from the origin security model and is stated here merely for clarity.

Many web browsers currently forbid by default any active mixed content on HTTPS pages. That is, when JavaScript is loaded from an HTTP origin onto an HTTPS page, an error is displayed and the HTTP content is not executed unless the user overrides the error. Any browser which enforces such a policy will also not permit access to WebRTC functionality from mixed content pages (because they never display mixed content). Browsers which allow active mixed content MUST nevertheless disable WebRTC functionality in mixed content settings.

Note that it is possible for a page which was not mixed content to become mixed content during the duration of the call. The major risk here is that the newly arrived insecure JS might redirect media to a location controlled by the attacker. Implementations MUST either choose to terminate the call or display a warning at that point.

Also note that the security architecture depends on the keying material not being available to move between origins. But, it is assumed that the identity assertion can be passed to anyone that the page cares to.

## 6.2. Device Permissions Model

Implementations MUST obtain explicit user consent prior to providing access to the camera and/or microphone. Implementations MUST at minimum support the following two permissions models for HTTPS origins.

- o Requests for one-time camera/microphone access.
- o Requests for permanent access.

Because HTTP origins cannot be securely established against network attackers, implementations MUST refuse all permissions grants for HTTP origins.

In addition, they SHOULD support requests for access that promise that media from this grant will be sent to a single communicating peer (obviously there could be other requests for other peers), eE.g., "Call customerservice@example.org". The semantics of this request are that the media stream from the camera and microphone will only be routed through a connection which has been cryptographically verified (through the IdP mechanism or an X.509 certificate in the DTLS-SRTP handshake) as being associated with the stated identity. Note that it is unlikely that browsers would have X.509 certificates, but servers might. Browsers servicing such requests SHOULD clearly indicate that identity to the user when asking for permission. The idea behind this type of permissions is that a user might have a

fairly narrow list of peers he is willing to communicate with, e.g., "my mother" rather than "anyone on Facebook". Narrow permissions grants allow the browser to do that enforcement.

**API Requirement:** The API **MUST** provide a mechanism for the requesting JS to relinquish the ability to see or modify the media (e.g., via `MediaStream.record()`). Combined with secure authentication of the communicating peer, this allows a user to be sure that the calling site is not accessing or modifying their conversion.

**UI Requirement:** The UI **MUST** clearly indicate when the user's camera and microphone are in use. This indication **MUST NOT** be suppressable by the JS and **MUST** clearly indicate how to terminate device access, and provide a UI means to immediately stop camera/microphone input without the JS being able to prevent it.

**UI Requirement:** If the UI indication of camera/microphone use are displayed in the browser such that minimizing the browser window would hide the indication, or the JS creating an overlapping window would hide the indication, then the browser **SHOULD** stop camera and microphone input when the indication is hidden. [Note: this may not be necessary in systems that are non-windows-based but that have good notifications support, such as phones.]

- o Browsers **MUST NOT** permit permanent screen or application sharing permissions to be installed as a response to a JS request for permissions. Instead, they must require some other user action such as a permissions setting or an application install experience to grant permission to a site.
- o Browsers **MUST** provide a separate dialog request for screen/application sharing permissions even if the media request is made at the same time as camera and microphone.
- o The browser **MUST** indicate any windows which are currently being shared in some unambiguous way. Windows which are not visible **MUST NOT** be shared even if the application is being shared. If the screen is being shared, then that **MUST** be indicated.

Browsers **MAY** permit the formation of data channels without any direct user approval. Because sites can always tunnel data through the server, further restrictions on the data channel do not provide any additional security. (See Section 6.3 for a related issue).

Implementations which support some form of direct user authentication **SHOULD** also provide a policy by which a user can authorize calls only to specific communicating peers. Specifically, the implementation **SHOULD** provide the following interfaces/controls:

- o Allow future calls to this verified user.
- o Allow future calls to any verified user who is in my system address book (this only works with address book integration, of course).

Implementations SHOULD also provide a different user interface indication when calls are in progress to users whose identities are directly verifiable. Section 6.5 provides more on this.

### 6.3. Communications Consent

Browser client implementations of WebRTC MUST implement ICE. Server gateway implementations which operate only at public IP addresses MUST implement either full ICE or ICE-Lite [RFC8445].

Browser implementations MUST verify reachability via ICE prior to sending any non-ICE packets to a given destination. Implementations MUST NOT provide the ICE transaction ID to JavaScript during the lifetime of the transaction (i.e., during the period when the ICE stack would accept a new response for that transaction). The JS MUST NOT be permitted to control the local ufrag and password, though it of course knows it.

While continuing consent is required, the ICE [RFC8445]; Section 10 keepalives use STUN Binding Indications which are one-way and therefore not sufficient. The current WG consensus is to use ICE Binding Requests for continuing consent freshness. ICE already requires that implementations respond to such requests, so this approach is maximally compatible. A separate document will profile the ICE timers to be used; see [RFC7675].

### 6.4. IP Location Privacy

A side effect of the default ICE behavior is that the peer learns one's IP address, which leaks large amounts of location information. This has negative privacy consequences in some circumstances. The API requirements in this section are intended to mitigate this issue. Note that these requirements are not intended to protect the user's IP address from a malicious site. In general, the site will learn at least a user's server reflexive address from any HTTP transaction. Rather, these requirements are intended to allow a site to cooperate with the user to hide the user's IP address from the other side of the call. Hiding the user's IP address from the server requires some sort of explicit privacy preserving mechanism on the client (e.g., Tor Browser [<https://www.torproject.org/projects/torbrowser.html.en>]) and is out of scope for this specification.

API Requirement: The API MUST provide a mechanism to allow the JS to suppress ICE negotiation (though perhaps to allow candidate gathering) until the user has decided to answer the call [note: determining when the call has been answered is a question for the JS.] This enables a user to prevent a peer from learning their IP address if they elect not to answer a call and also from learning whether the user is online.

API Requirement: The API MUST provide a mechanism for the calling application JS to indicate that only TURN candidates are to be used. This prevents the peer from learning one's IP address at all. This mechanism MUST also permit suppression of the related address field, since that leaks local addresses.

API Requirement: The API MUST provide a mechanism for the calling application to reconfigure an existing call to add non-TURN candidates. Taken together, this and the previous requirement allow ICE negotiation to start immediately on incoming call notification, thus reducing post-dial delay, but also to avoid disclosing the user's IP address until they have decided to answer. They also allow users to completely hide their IP address for the duration of the call. Finally, they allow a mechanism for the user to optimize performance by reconfiguring to allow non-TURN candidates during an active call if the user decides they no longer need to hide their IP address

Note that some enterprises may operate proxies and/or NATs designed to hide internal IP addresses from the outside world. WebRTC provides no explicit mechanism to allow this function. Either such enterprises need to proxy the HTTP/HTTPS and modify the SDP and/or the JS, or there needs to be browser support to set the "TURN-only" policy regardless of the site's preferences.

## 6.5. Communications Security

Implementations MUST support SRTP [RFC3711]. Implementations MUST support DTLS [RFC6347] and DTLS-SRTP [RFC5763][RFC5764] for SRTP keying. Implementations MUST support SCTP over DTLS [RFC8261].

All media channels MUST be secured via SRTP and SRTCP. Media traffic MUST NOT be sent over plain (unencrypted) RTP or RTCP; that is, implementations MUST NOT negotiate cipher suites with NULL encryption modes. DTLS-SRTP MUST be offered for every media channel. WebRTC implementations MUST NOT offer SDP Security Descriptions [RFC4568] or select it if offered. A SRTP MKI MUST NOT be used.

All data channels MUST be secured via DTLS.

All Implementations MUST support DTLS 1.2 with the TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 cipher suite and the P-256 curve [FIPS186]. Earlier drafts of this specification required DTLS 1.0 with the cipher suite TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA, and at the time of this writing some implementations do not support DTLS 1.2; endpoints which support only DTLS 1.2 might encounter interoperability issues. The DTLS-SRTP protection profile SRTP\_AES128\_CM\_HMAC\_SHA1\_80 MUST be supported for SRTP. Implementations MUST favor cipher suites which support (Perfect Forward Secrecy) PFS over non-PFS cipher suites and SHOULD favor AEAD over non-AEAD cipher suites.

Implementations MUST NOT implement DTLS renegotiation and MUST reject it with a "no\_renegotiation" alert if offered.

Endpoints MUST NOT implement TLS False Start [RFC7918].

API Requirement: The API MUST generate a new authentication key pair for every new call by default. This is intended to allow for unlinkability.

API Requirement: The API MUST provide a means to reuse a key pair for calls. This can be used to enable key continuity-based authentication, and could be used to amortize key generation costs.

API Requirement: Unless the user specifically configures an external key pair, different key pairs MUST be used for each origin. (This avoids creating a super-cookie.)

API Requirement: When DTLS-SRTP is used, the API MUST NOT permit the JS to obtain the negotiated keying material. This requirement preserves the end-to-end security of the media.

UI Requirements: A user-oriented client MUST provide an "inspector" interface which allows the user to determine the security characteristics of the media.

The following properties SHOULD be displayed "up-front" in the browser chrome, i.e., without requiring the user to ask for them:

- \* A client MUST provide a user interface through which a user may determine the security characteristics for currently-displayed audio and video stream(s)

- \* A client MUST provide a user interface through which a user may determine the security characteristics for transmissions of their microphone audio and camera video.
- \* If the far endpoint was directly verified, either via a third-party verifiable X.509 certificate or via a Web IdP mechanism (see Section 7) the "security characteristics" MUST include the verified information. X.509 identities and Web IdP identities have similar semantics and should be displayed in a similar way.

The following properties are more likely to require some "drill-down" from the user:

- \* The "security characteristics" MUST indicate the cryptographic algorithms in use (For example: "AES-CBC".)
- \* The "security characteristics" MUST indicate whether PFS is provided.
- \* The "security characteristics" MUST include some mechanism to allow an out-of-band verification of the peer, such as a certificate fingerprint or a Short Authentication String (SAS). These are compared by the peers to authenticate one another.

## 7. Web-Based Peer Authentication

In a number of cases, it is desirable for the endpoint (i.e., the browser) to be able to directly identify the endpoint on the other side without trusting the signaling service to which they are connected. For instance, users may be making a call via a federated system where they wish to get direct authentication of the other side. Alternately, they may be making a call on a site which they minimally trust (such as a poker site) but to someone who has an identity on a site they do trust (such as a social network.)

Recently, a number of Web-based identity technologies (OAuth, Facebook Connect etc.) have been developed. While the details vary, what these technologies share is that they have a Web-based (i.e., HTTP/HTTPS) identity provider which attests to Alice's identity. For instance, if Alice has an account at example.org, Alice could use the example.org identity provider to prove to others that Alice is alice@example.org. The development of these technologies allows us

to separate calling from identity provision: Alice could call you on a poker site but identify herself as `alice@example.org`.

Whatever the underlying technology, the general principle is that the party which is being authenticated is NOT the signaling site but rather the user (and their browser). Similarly, the relying party is the browser and not the signaling site. Thus, the browser **MUST** generate the input to the IdP assertion process and display the results of the verification process to the user in a way which cannot be imitated by the calling site.

The mechanisms defined in this document do not require the browser to implement any particular identity protocol or to support any particular IdP. Instead, this document provides a generic interface which any IdP can implement. Thus, new IdPs and protocols can be introduced without change to either the browser or the calling service. This avoids the need to make a commitment to any particular identity protocol, although browsers may opt to directly implement some identity protocols in order to provide superior performance or UI properties.

#### 7.1. Trust Relationships: IdPs, APs, and RPs

Any federated identity protocol has three major participants:

**Authenticating Party (AP):** The entity which is trying to establish its identity.

**Identity Provider (IdP):** The entity which is vouching for the AP's identity.

**Relying Party (RP):** The entity which is trying to verify the AP's identity.

The AP and the IdP have an account relationship of some kind: the AP registers with the IdP and is able to subsequently authenticate directly to the IdP (e.g., with a password). This means that the browser must somehow know which IdP(s) the user has an account relationship with. This can either be something that the user configures into the browser or that is configured at the calling site and then provided to the PeerConnection by the Web application at the calling site. The use case for having this information configured into the browser is that the user may "log into" the browser to bind it to some identity. This is becoming common in new browsers.

However, it should also be possible for the IdP information to simply be provided by the calling application.

At a high level there are two kinds of IdPs:

**Authoritative:** IdPs which have verifiable control of some section of the identity space. For instance, in the realm of e-mail, the operator of "example.com" has complete control of the namespace ending in "@example.com". Thus, "alice@example.com" is whoever the operator says it is. Examples of systems with authoritative identity providers include DNSSEC, RFC 4474, and Facebook Connect (Facebook identities only make sense within the context of the Facebook system).

**Third-Party:** IdPs which don't have control of their section of the identity space but instead verify user's identities via some unspecified mechanism and then attest to it. Because the IdP doesn't actually control the namespace, RPs need to trust that the IdP is correctly verifying AP identities, and there can potentially be multiple IdPs attesting to the same section of the identity space. Probably the best-known example of a third-party identity provider is SSL/TLS certificates, where there are a large number of CAs all of whom can attest to any domain name.

If an AP is authenticating via an authoritative IdP, then the RP does not need to explicitly configure trust in the IdP at all. The identity mechanism can directly verify that the IdP indeed made the relevant identity assertion (a function provided by the mechanisms in this document), and any assertion it makes about an identity for which it is authoritative is directly verifiable. Note that this does not mean that the IdP might not lie, but that is a trustworthiness judgement that the user can make at the time he looks at the identity.

By contrast, if an AP is authenticating via a third-party IdP, the RP needs to explicitly trust that IdP (hence the need for an explicit trust anchor list in PKI-based SSL/TLS clients). The list of trustable IdPs needs to be configured directly into the browser, either by the user or potentially by the browser manufacturer. This is a significant advantage of authoritative IdPs and implies that if third-party IdPs are to be supported, the potential number needs to be fairly small.

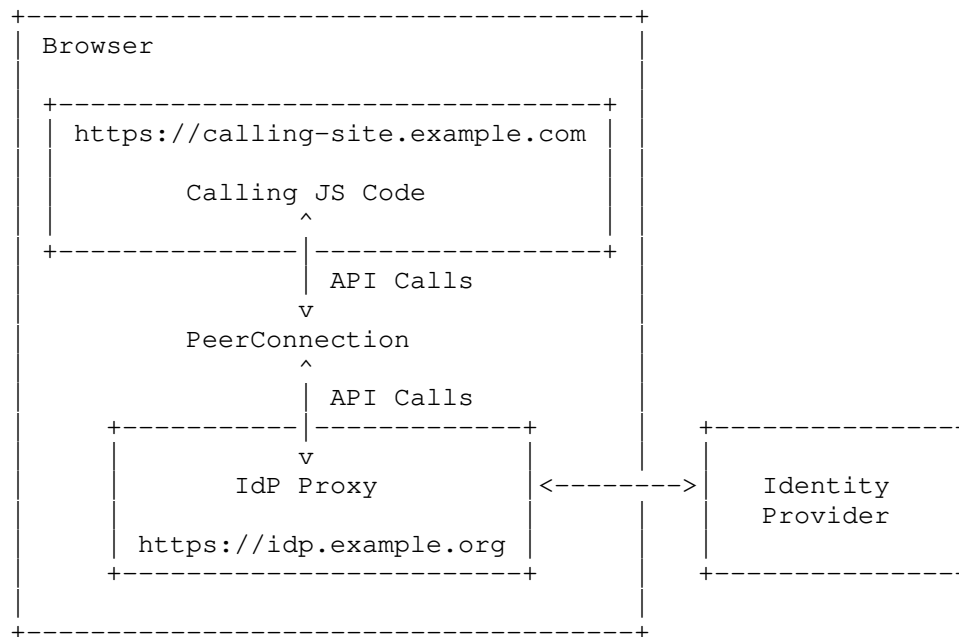
## 7.2. Overview of Operation

In order to provide security without trusting the calling site, the `PeerConnection` component of the browser must interact directly with the IdP. The details of the mechanism are described in the W3C API specification, but the general idea is that the `PeerConnection` component downloads JS from a specific location on the IdP dictated by the IdP domain name. That JS (the "IdP proxy") runs in an isolated security context within the browser and the `PeerConnection` talks to it via a secure message passing channel.

Note that there are two logically separate functions here:

- o Identity assertion generation.
- o Identity assertion verification.

The same IdP JS "endpoint" is used for both functions but of course a given IdP might behave differently and load new JS to perform one function or the other.



When the `PeerConnection` object wants to interact with the IdP, the sequence of events is as follows:

1. The browser (the PeerConnection component) instantiates an IdP proxy. This allows the IdP to load whatever JS is necessary into the proxy. The resulting code runs in the IdP's security context.
2. The IdP registers an object with the browser that conforms to the API defined in [webrtc-api].
3. The browser invokes methods on the object registered by the IdP proxy to create or verify identity assertions.

This approach allows us to decouple the browser from any particular identity provider; the browser need only know how to load the IdP's JavaScript--the location of which is determined based on the IdP's identity--and to call the generic API for requesting and verifying identity assertions. The IdP provides whatever logic is necessary to bridge the generic protocol to the IdP's specific requirements. Thus, a single browser can support any number of identity protocols, including being forward compatible with IdPs which did not exist at the time the browser was written.

### 7.3. Items for Standardization

There are two parts to this work:

- o The precise information from the signaling message that must be cryptographically bound to the user's identity and a mechanism for carrying assertions in JSEP messages. This is specified in Section 7.4.
- o The interface to the IdP, which is defined in the companion W3C WebRTC API specification [webrtc-api].

The WebRTC API specification also defines JavaScript interfaces that the calling application can use to specify which IdP to use. That API also provides access to the assertion-generation capability and the status of the validation process.

### 7.4. Binding Identity Assertions to JSEP Offer/Answer Transactions

An identity assertion binds the user's identity (as asserted by the IdP) to the SDP offer/answer exchange and specifically to the media. In order to achieve this, the PeerConnection must provide the DTLS-SRTP fingerprint to be bound to the identity. This is provided as a JavaScript object (also known as a dictionary or hash) with a single "fingerprint" key, as shown below:

```
{
  "fingerprint":
  [
    { "algorithm": "sha-256",
      "digest": "4A:AD:B9:B1:3F:....:E5:7C:AB" },
    { "algorithm": "sha-1",
      "digest": "74:E9:76:C8:19:....:F4:45:6B" }
  ]
}
```

The "fingerprint" value is an array of objects. Each object in the array contains "algorithm" and "digest" values, which correspond directly to the algorithm and digest values in the "fingerprint" attribute of the SDP [RFC8122].

This object is encoded in a JSON [RFC8259] string for passing to the IdP. The identity assertion returned by the IdP, which is encoded in the "identity" attribute, is a JSON object that is encoded as described in Section 7.4.1.

This structure does not need to be interpreted by the IdP or the IdP proxy. It is consumed solely by the RP's browser. The IdP merely treats it as an opaque value to be attested to. Thus, new parameters can be added to the assertion without modifying the IdP.

#### 7.4.1. Carrying Identity Assertions

Once an IdP has generated an assertion (see Section 7.6), it is attached to the SDP offer/answer message. This is done by adding a new 'identity' attribute to the SDP. The sole contents of this value is the identity assertion. The identity assertion produced by the IdP is encoded into a UTF-8 JSON text, then Base64-encoded [RFC4648] to produce this string. For example:

```

v=0
o=- 1181923068 1181923196 IN IP4 ual.example.com
s=example1
c=IN IP4 ual.example.com
a=fingerprint:sha-1 \
  4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=identity:\
  eyJpZHAiOnsizG9tYWluIjoizXhhbXBsZS5vcmdiLCJwcm90b2NvbCI6ImJvZ3Vz\
  In0sImFzc2VydgIvbiI6IntcImlkZW50aXR5XCI6XCJib2JAZXhhbXBsZS5vcmdc\
  IixcImNvbmlbnRzXCI6XCJhYmNkZWZnaGlqa2xtbm9wcXJzdHV2d3l6XCIsXCJz\
  aWduYXR1cmVcIjpcIjAxMDIwMzA0MDUwNlwiJ9
a=...
t=0 0
m=audio 6056 RTP/SAVP 0
a=sendrecv
...

```

Note that long lines in the example are folded to meet the column width constraints of this document; the backslash ("`\`") at the end of a line, the carriage return that follows, and whitespace shall be ignored.

The 'identity' attribute attests to all "fingerprint" attributes in the session description. It is therefore a session-level attribute.

Multiple "fingerprint" values can be used to offer alternative certificates for a peer. The "identity" attribute MUST include all fingerprint values that are included in "fingerprint" attributes of the session description.

The RP browser MUST verify that the in-use certificate for a DTLS connection is in the set of fingerprints returned from the IdP when verifying an assertion.

## 7.5. Determining the IdP URI

In order to ensure that the IdP is under control of the domain owner rather than someone who merely has an account on the domain owner's server (e.g., in shared hosting scenarios), the IdP JavaScript is hosted at a deterministic location based on the IdP's domain name. Each IdP proxy instance is associated with two values:

**Authority:** The authority [RFC3986] at which the IdP's service is hosted.

**protocol:** The specific IdP protocol which the IdP is using. This is a completely opaque IdP-specific string, but allows an IdP to implement two protocols in parallel. This value may be the empty

string. If no value for protocol is provided, a value of "default" is used.

Each IdP MUST serve its initial entry page (i.e., the one loaded by the IdP proxy) from a well-known URI [RFC5785]. The well-known URI for an IdP proxy is formed from the following URI components:

1. The scheme, "https:". An IdP MUST be loaded using HTTPS [RFC2818].
2. The authority [RFC3986]. As noted above, the authority MAY contain a non-default port number or userinfo sub-component. Both are removed when determining if an asserted identity matches the name of the IdP.
3. The path, starting with "/.well-known/idp-proxy/" and appended with the IdP protocol. Note that the separator characters '/' (%2F) and '\' (%5C) MUST NOT be permitted in the protocol field, lest an attacker be able to direct requests outside of the controlled "/.well-known/" prefix. Query and fragment values MAY be used by including '?' or '#' characters.

For example, for the IdP "identity.example.com" and the protocol "example", the URL would be:

```
https://identity.example.com/.well-known/idp-proxy/example
```

The IdP MAY redirect requests to this URL, but they MUST retain the "https" scheme. This changes the effective origin of the IdP, but not the domain of the identities that the IdP is permitted to assert and validate. I.e., the IdP is still regarded as authoritative for the original domain.

#### 7.5.1. Authenticating Party

How an AP determines the appropriate IdP domain is out of scope of this specification. In general, however, the AP has some actual account relationship with the IdP, as this identity is what the IdP is attesting to. Thus, the AP somehow supplies the IdP information to the browser. Some potential mechanisms include:

- o Provided by the user directly.
- o Selected from some set of IdPs known to the calling site. E.g., a button that shows "Authenticate via Facebook Connect"

### 7.5.2. Relying Party

Unlike the AP, the RP need not have any particular relationship with the IdP. Rather, it needs to be able to process whatever assertion is provided by the AP. As the assertion contains the IdP's identity in the "idp" field of the JSON-encoded object (see Section 7.6), the URI can be constructed directly from the assertion, and thus the RP can directly verify the technical validity of the assertion with no user interaction. Authoritative assertions need only be verifiable. Third-party assertions also MUST be verified against local policy, as described in Section 8.1.

### 7.6. Requesting Assertions

The input to identity assertion is the JSON-encoded object described in Section 7.4 that contains the set of certificate fingerprints the browser intends to use. This string is treated as opaque from the perspective of the IdP.

The browser also identifies the origin that the PeerConnection is run in, which allows the IdP to make decisions based on who is requesting the assertion.

An application can optionally provide a user identifier hint when specifying an IdP. This value is a hint that the IdP can use to select amongst multiple identities, or to avoid providing assertions for unwanted identities. The "username" is a string that has no meaning to any entity other than the IdP, it can contain any data the IdP needs in order to correctly generate an assertion.

An identity assertion that is successfully provided by the IdP consists of the following information:

idp: The domain name of an IdP and the protocol string. This MAY identify a different IdP or protocol from the one that generated the assertion.

assertion: An opaque value containing the assertion itself. This is only interpretable by the identified IdP or the IdP code running in the client.

Figure 5 shows an example assertion formatted as JSON. In this case, the message has presumably been digitally signed/MACed in some way that the IdP can later verify it, but this is an implementation detail and out of scope of this document.

```
{
  "idp":{
    "domain": "example.org",
    "protocol": "bogus"
  },
  "assertion": "{\\"identity\\":\\"bob@example.org\\",
                \\"contents\\":\\"abcdefghijklmnopqrstuvwyz\\",
                \\"signature\\":\\"010203040506\\"}"
}
```

Figure 5: Example assertion

For use in signaling, the assertion is serialized into JSON, Base64-encoded [RFC4648], and used as the value of the "identity" attribute. IdPs SHOULD ensure that any assertions they generate cannot be interpreted in a different context. E.g., they should use a distinct format or have separate cryptographic keys for assertion generation and other purposes. Line breaks are inserted solely for readability.

#### 7.7. Managing User Login

In order to generate an identity assertion, the IdP needs proof of the user's identity. It is common practice to authenticate users (using passwords or multi-factor authentication), then use Cookies [RFC6265] or HTTP authentication [RFC7617] for subsequent exchanges.

The IdP proxy is able to access cookies, HTTP authentication or other persistent session data because it operates in the security context of the IdP origin. Therefore, if a user is logged in, the IdP could have all the information needed to generate an assertion.

An IdP proxy is unable to generate an assertion if the user is not logged in, or the IdP wants to interact with the user to acquire more information before generating the assertion. If the IdP wants to interact with the user before generating an assertion, the IdP proxy can fail to generate an assertion and instead indicate a URL where login should proceed.

The application can then load the provided URL to enable the user to enter credentials. The communication between the application and the IdP is described in [webrtc-api].

#### 8. Verifying Assertions

The input to identity validation is the assertion string taken from a decoded 'identity' attribute.

The IdP proxy verifies the assertion. Depending on the identity protocol, the proxy might contact the IdP server or other servers. For instance, an OAuth-based protocol will likely require using the IdP as an oracle, whereas with a signature-based scheme might be able to verify the assertion without contacting the IdP, provided that it has cached the relevant public key.

Regardless of the mechanism, if verification succeeds, a successful response from the IdP proxy consists of the following information:

identity: The identity of the AP from the IdP's perspective.  
Details of this are provided in Section 8.1.

contents: The original unmodified string provided by the AP as input to the assertion generation process.

Figure 6 shows an example response, which is JSON-formatted.

```
{
  "identity": "bob@example.org",
  "contents": "{\"fingerprint\":[ ... ]}"
}
```

Figure 6: Example verification result

### 8.1. Identity Formats

The identity provided from the IdP to the RP browser MUST consist of a string representing the user's identity. This string is in the form "<user>@<domain>", where "user" consists of any character, and domain is an internationalized domain name [RFC5890] encoded as a sequence of U-labels.

The PeerConnection API MUST check this string as follows:

1. If the "domain" portion of the string is equal to the domain name of the IdP proxy, then the assertion is valid, as the IdP is authoritative for this domain. Comparison of domain names is done using the label equivalence rule defined in Section 2.3.2.4 of [RFC5890].
2. If the "domain" portion of the string is not equal to the domain name of the IdP proxy, then the PeerConnection object MUST reject the assertion unless both:
  1. the IdP domain is trusted as an acceptable third-party IdP;  
and

2. local policy is configured to trust this IdP domain for the domain portion of the identity string.

Any "@" or "%" characters in the "user" portion of the identity MUST be escaped according to the "Percent-Encoding" rules defined in Section 2.1 of [RFC3986]. Characters other than "@" and "%" MUST NOT be percent-encoded. For example, with a "user" of "user@133" and a "domain" of "identity.example.com", the resulting string will be encoded as "user%40133@identity.example.com".

Implementations are cautioned to take care when displaying user identities containing escaped "@" characters. If such characters are unescaped prior to display, implementations MUST distinguish between the domain of the IdP proxy and any domain that might be implied by the portion of the "<user>" portion that appears after the escaped "@" sign.

## 9. Security Considerations

Much of the security analysis of this problem is contained in [I-D.ietf-rtcweb-security] or in the discussion of the particular issues above. In order to avoid repetition, this section focuses on (a) residual threats that are not addressed by this document and (b) threats produced by failure/misbehavior of one of the components in the system.

### 9.1. Communications Security

IF HTTPS is not used to secure communications to the signaling server, and the identity mechanism used in Section 7 is not used, then any on-path attacker can replace the DTLS-SRTP fingerprints in the handshake and thus substitute its own identity for that of either endpoint.

Even if HTTPS is used, the signaling server can potentially mount a man-in-the-middle attack unless implementations have some mechanism for independently verifying keys. The UI requirements in Section 6.5 are designed to provide such a mechanism for motivated/security conscious users, but are not suitable for general use. The identity service mechanisms in Section 7 are more suitable for general use. Note, however, that a malicious signaling service can strip off any such identity assertions, though it cannot forge new ones. Note that all of the third-party security mechanisms available (whether X.509 certificates or a third-party IdP) rely on the security of the third party--this is of course also true of the user's connection to the Web site itself. Users who wish to assure themselves of security against a malicious identity provider can only do so by verifying

peer credentials directly, e.g., by checking the peer's fingerprint against a value delivered out of band.

In order to protect against malicious content JavaScript, that JavaScript MUST NOT be allowed to have direct access to---or perform computations with---DTLS keys. For instance, if content JS were able to compute digital signatures, then it would be possible for content JS to get an identity assertion for a browser's generated key and then use that assertion plus a signature by the key to authenticate a call protected under an ephemeral Diffie-Hellman (DH) key controlled by the content JS, thus violating the security guarantees otherwise provided by the IdP mechanism. Note that it is not sufficient merely to deny the content JS direct access to the keys, as some have suggested doing with the WebCrypto API [webcrypto]. The JS must also not be allowed to perform operations that would be valid for a DTLS endpoint. By far the safest approach is simply to deny the ability to perform any operations that depend on secret information associated with the key. Operations that depend on public information, such as exporting the public key are of course safe.

## 9.2. Privacy

The requirements in this document are intended to allow:

- o Users to participate in calls without revealing their location.
- o Potential callees to avoid revealing their location and even presence status prior to agreeing to answer a call.

However, these privacy protections come at a performance cost in terms of using TURN relays and, in the latter case, delaying ICE. Sites SHOULD make users aware of these tradeoffs.

Note that the protections provided here assume a non-malicious calling service. As the calling service always knows the users status and (absent the use of a technology like Tor) their IP address, they can violate the users privacy at will. Users who wish privacy against the calling sites they are using must use separate privacy enhancing technologies such as Tor. Combined WebRTC/Tor implementations SHOULD arrange to route the media as well as the signaling through Tor. Currently this will produce very suboptimal performance.

Additionally, any identifier which persists across multiple calls is potentially a problem for privacy, especially for anonymous calling services. Such services SHOULD instruct the browser to use separate DTLS keys for each call and also to use TURN throughout the call. Otherwise, the other side will learn linkable information that would

allow them to correlate the browser across multiple calls. Additionally, browsers SHOULD implement the privacy-preserving CNAME generation mode of [RFC7022].

### 9.3. Denial of Service

The consent mechanisms described in this document are intended to mitigate denial of service attacks in which an attacker uses clients to send large amounts of traffic to a victim without the consent of the victim. While these mechanisms are sufficient to protect victims who have not implemented WebRTC at all, WebRTC implementations need to be more careful.

Consider the case of a call center which accepts calls via WebRTC. An attacker proxies the call center's front-end and arranges for multiple clients to initiate calls to the call center. Note that this requires user consent in many cases but because the data channel does not need consent, he can use that directly. Since ICE will complete, browsers can then be induced to send large amounts of data to the victim call center if it supports the data channel at all. Preventing this attack requires that automated WebRTC implementations implement sensible flow control and have the ability to triage out (i.e., stop responding to ICE probes on) calls which are behaving badly, and especially to be prepared to remotely throttle the data channel in the absence of plausible audio and video (which the attacker cannot control).

Another related attack is for the signaling service to swap the ICE candidates for the audio and video streams, thus forcing a browser to send video to the sink that the other victim expects will contain audio (perhaps it is only expecting audio!) potentially causing overload. Muxing multiple media flows over a single transport makes it harder to individually suppress a single flow by denying ICE keepalives. Either media-level (RTCP) mechanisms must be used or the implementation must deny responses entirely, thus terminating the call.

Yet another attack, suggested by Magnus Westerlund, is for the attacker to cross-connect offers and answers as follows. It induces the victim to make a call and then uses its control of other users' browsers to get them to attempt a call to someone. It then translates their offers into apparent answers to the victim, which looks like large-scale parallel forking. The victim still responds to ICE responses and now the browsers all try to send media to the victim. Implementations can defend themselves from this attack by only responding to ICE Binding Requests for a limited number of remote ufrags (this is the reason for the requirement that the JS not be able to control the ufrag and password).

[I-D.ietf-rtcweb-rtp-usage] Section 13 documents a number of potential RTCP-based DoS attacks and countermeasures.

Note that attacks based on confusing one end or the other about consent are possible even in the face of the third-party identity mechanism as long as major parts of the signaling messages are not signed. On the other hand, signing the entire message severely restricts the capabilities of the calling application, so there are difficult tradeoffs here.

#### 9.4. IdP Authentication Mechanism

This mechanism relies for its security on the IdP and on the PeerConnection correctly enforcing the security invariants described above. At a high level, the IdP is attesting that the user identified in the assertion wishes to be associated with the assertion. Thus, it must not be possible for arbitrary third parties to get assertions tied to a user or to produce assertions that RPs will accept.

##### 9.4.1. PeerConnection Origin Check

Fundamentally, the IdP proxy is just a piece of HTML and JS loaded by the browser, so nothing stops a Web attacker from creating their own IFRAME, loading the IdP proxy HTML/JS, and requesting a signature over his own keys rather than those generated in the browser. However, that proxy would be in the attacker's origin, not the IdP's origin. Only the browser itself can instantiate a context that (a) is in the IdP's origin and (b) exposes the correct API surface. Thus, the IdP proxy on the sender's side MUST ensure that it is running in the IdP's origin prior to issuing assertions.

Note that this check only asserts that the browser (or some other entity with access to the user's authentication data) attests to the request and hence to the fingerprint. It does not demonstrate that the browser has access to the associated private key, and therefore an attacker can attach their own identity to another party's keying material, thus making a call which comes from Alice appear to come from the attacker. See [I-D.ietf-mmusic-sdp-uks] for defenses against this form of attack.

##### 9.4.2. IdP Well-known URI

As described in Section 7.5 the IdP proxy HTML/JS landing page is located at a well-known URI based on the IdP's domain name. This requirement prevents an attacker who can write some resources at the IdP (e.g., on one's Facebook wall) from being able to impersonate the IdP.

#### 9.4.3. Privacy of IdP-generated identities and the hosting site

Depending on the structure of the IdP's assertions, the calling site may learn the user's identity from the perspective of the IdP. In many cases this is not an issue because the user is authenticating to the site via the IdP in any case, for instance when the user has logged in with Facebook Connect and is then authenticating their call with a Facebook identity. However, in other case, the user may not have already revealed their identity to the site. In general, IdPs SHOULD either verify that the user is willing to have their identity revealed to the site (e.g., through the usual IdP permissions dialog) or arrange that the identity information is only available to known RPs (e.g., social graph adjacencies) but not to the calling site. The "domain" field of the assertion request can be used to check that the user has agreed to disclose their identity to the calling site; because it is supplied by the PeerConnection it can be trusted to be correct.

#### 9.4.4. Security of Third-Party IdPs

As discussed above, each third-party IdP represents a new universal trust point and therefore the number of these IdPs needs to be quite limited. Most IdPs, even those which issue unqualified identities such as Facebook, can be recast as authoritative IdPs (e.g., 123456@facebook.com). However, in such cases, the user interface implications are not entirely desirable. One intermediate approach is to have special (potentially user configurable) UI for large authoritative IdPs, thus allowing the user to instantly grasp that the call is being authenticated by Facebook, Google, etc.

##### 9.4.4.1. Confusable Characters

Because a broad range of characters are permitted in identity strings, it may be possible for attackers to craft identities which are confusable with other identities (see [RFC6943] for more on this topic). This is a problem with any identifier space of this type (e.g., e-mail addresses). Those minting identifiers should avoid mixed scripts and similar confusable characters. Those presenting these identifiers to a user should consider highlighting cases of mixed script usage (see [RFC5890], section 4.4). Other best practices are still in development.

#### 9.4.5. Web Security Feature Interactions

A number of optional Web security features have the potential to cause issues for this mechanism, as discussed below.

#### 9.4.5.1. Popup Blocking

When popup blocking is in use, the IdP proxy is unable to generate popup windows, dialogs or any other form of user interactions. This prevents the IdP proxy from being used to circumvent user interaction. The "LOGINNEEDED" message allows the IdP proxy to inform the calling site of a need for user login, providing the information necessary to satisfy this requirement without resorting to direct user interaction from the IdP proxy itself.

#### 9.4.5.2. Third Party Cookies

Some browsers allow users to block third party cookies (cookies associated with origins other than the top level page) for privacy reasons. Any IdP which uses cookies to persist logins will be broken by third-party cookie blocking. One option is to accept this as a limitation; another is to have the PeerConnection object disable third-party cookie blocking for the IdP proxy.

### 10. IANA Considerations

This specification defines the "identity" SDP attribute per the procedures of Section 8.2.4 of [RFC4566]. The required information for the registration is included here:

Contact Name: IESG (iesg@ietf.org)

Attribute Name: identity

Long Form: identity

Type of Attribute: session-level

Charset Considerations: This attribute is not subject to the charset attribute.

Purpose: This attribute carries an identity assertion, binding an identity to the transport-level security session.

Appropriate Values: See Section 5 of RFCXXXX [[Editor Note: This document.]]

Mux Category: NORMAL.

This section registers the "idp-proxy" well-known URI from [RFC5785].

URI suffix: idp-proxy

Change controller: IETF

## 11. Acknowledgements

Bernard Aboba, Harald Alvestrand, Richard Barnes, Dan Druta, Cullen Jennings, Hadriel Kaplan, Matthew Kaufman, Jim McEachern, Martin Thomson, Magnus Westerland. Matthew Kaufman provided the UI material in Section 6.5. Christer Holmberg provided the initial version of Section 5.1.

## 12. Changes

[RFC Editor: Please remove this section prior to publication.]

### 12.1. Changes since -15

Rewrite the Identity section in more conventional offer/answer format.

Clarify rules on changing identities.

### 12.2. Changes since -11

Update discussion of IdP security model

Replace "domain name" with RFC 3986 Authority

Clean up discussion of how to generate IdP URI.

Remove obsolete text about null cipher suites.

Remove obsolete appendixes about older IdP systems

Require support for ECDSA, PFS, and AEAD

### 12.3. Changes since -10

Update cipher suite profiles.

Rework IdP interaction based on implementation experience in Firefox.

### 12.4. Changes since -06

Replaced RTCWEB and RTC-Web with WebRTC, except when referring to the IETF WG

Forbade use in mixed content as discussed in Orlando.

Added a requirement to surface NULL ciphers to the top-level.

Tried to clarify SRTP versus DTLS-SRTP.

Added a section on screen sharing permissions.

Assorted editorial work.

#### 12.5. Changes since -05

The following changes have been made since the -05 draft.

- o Response to comments from Richard Barnes
- o More explanation of the IdP security properties and the federation use case.
- o Editorial cleanup.

#### 12.6. Changes since -03

Version -04 was a version control mistake. Please ignore.

The following changes have been made since the -04 draft.

- o Move origin check from IdP to RP per discussion in YVR.
- o Clarified treatment of X.509-level identities.
- o Editorial cleanup.

#### 12.7. Changes since -03

#### 12.8. Changes since -02

The following changes have been made since the -02 draft.

- o Forbid persistent HTTP permissions.
- o Clarified the text in S 5.4 to clearly refer to requirements on the API to provide functionality to the site.
- o Fold in the IETF portion of draft-rescorla-rtcweb-generic-idp
- o Retarget the continuing consent section to assume Binding Requests
- o Added some more privacy and linkage text in various places.

- o Editorial improvements

## 13. References

### 13.1. Normative References

- [FIPS186] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", NIST PUB 186-4 , July 2013.
- [I-D.ietf-mmusic-sdp-uks]  
Thomson, M. and E. Rescorla, "Unknown Key Share Attacks on uses of TLS with the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-uks-06 (work in progress), July 2019.
- [I-D.ietf-rtcweb-jsep]  
Uberti, J., Jennings, C., and E. Rescorla, "JavaScript Session Establishment Protocol", draft-ietf-rtcweb-jsep-26 (work in progress), February 2019.
- [I-D.ietf-rtcweb-overview]  
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-19 (work in progress), November 2017.
- [I-D.ietf-rtcweb-rtp-usage]  
Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-26 (work in progress), March 2016.
- [I-D.ietf-rtcweb-security]  
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-12 (work in progress), July 2019.
- [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>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.

- [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/info/rfc3264>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.
- [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>>.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<https://www.rfc-editor.org/info/rfc4566>>.
- [RFC4568] Andreassen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <<https://www.rfc-editor.org/info/rfc4568>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <<https://www.rfc-editor.org/info/rfc5763>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<https://www.rfc-editor.org/info/rfc5764>>.

- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, DOI 10.17487/RFC7022, September 2013, <<https://www.rfc-editor.org/info/rfc7022>>.
- [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/info/rfc7675>>.
- [RFC7918] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <<https://www.rfc-editor.org/info/rfc7918>>.
- [RFC8122] Lennox, J. and C. Holmberg, "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 8122, DOI 10.17487/RFC8122, March 2017, <<https://www.rfc-editor.org/info/rfc8122>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

- [RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <<https://www.rfc-editor.org/info/rfc8261>>.
- [RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <<https://www.rfc-editor.org/info/rfc8445>>.
- [webcrypto] editors, W., "Web Cryptography API", June 2013.  
Available at <http://www.w3.org/TR/WebCryptoAPI/>
- [webrtc-api] editors, W., "WebRTC 1.0: Real-time Communication Between Browsers", October 2011.  
Available at <http://dev.w3.org/2011/webrtc/editor/webrtc.html>

### 13.2. 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/info/rfc3261>>.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <<https://www.rfc-editor.org/info/rfc6120>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <<https://www.rfc-editor.org/info/rfc6943>>.

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <<https://www.rfc-editor.org/info/rfc7617>>.

[XmlHttpRequest]  
van Kesteren, A., "XMLHttpRequest Level 2", January 2012.

Author's Address

Eric Rescorla  
RTFM, Inc.  
2064 Edgewood Drive  
Palo Alto, CA 94303  
USA

Phone: +1 650 678 2350  
Email: [ekr@rtfm.com](mailto:ekr@rtfm.com)

RTCWeb  
Internet-Draft  
Intended status: Standards Track  
Expires: January 10, 2013

M. Isomaki  
Nokia  
July 9, 2012

RTCweb Considerations for Mobile Devices  
draft-isomaki-rtcweb-mobile-00

Abstract

Web Real-time Communications (WebRTC) aims to provide web-based applications real-time and peer-to-peer communication capabilities. In many cases those applications are run in mobile devices connected to different types of mobile networks. This document gives an overview of the issues and challenges in implementing and deploying WebRTC in mobile environments. It also gives guidance on how to overcome those challenges.

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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 10, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Common mobile networks and their properties . . . . .	3
3. Specific issues and how to deal with them . . . . .	5
3.1. Persistent connectivity to the Calling Site . . . . .	5
3.2. Media and Data channels . . . . .	6
3.3. Recovery from interface switching . . . . .	7
3.4. Congestion avoidance . . . . .	9
4. Security Considerations . . . . .	9
5. Acknowledgements . . . . .	9
6. References . . . . .	10
Author's Address . . . . .	10

## 1. Introduction

Web Real-time Communications (WebRTC) provides web-based applications real-time and peer-to-peer communication capabilities. The applications can setup communication sessions that can carry audio, video, or any application specific data. To be reachable for incoming sessions setups or other messages, the applications must keep persistent connectivity with their "calling site".

In the last few years, mobile devices, such as smartphones or tablets, have become relatively powerful in terms of processing and memory. Their browsers are becoming close to their desktop counterparts. So, from that perspective, it is feasible to run WebRTC applications in them. However, power consumption and highly diverse nature of the connectivity still remain as specific challenges. A lot of work is done to address these challenges in e.g. radio technologies and hardware components, but still by far the most important factor is how the applications and protocols and application programming interfaces are designed.

Section 2 of this document gives an overview of the characteristics of different mobile networks as background for further discussion. Section 3 introduces the specific issues that WebRTC protocols and applications should take into consideration to be mobile-friendly.

The current version of the document misses all references and lot of details. It may have some errors. Its purpose is to get attention to the topics it raises and start discussion about them.

## 2. Common mobile networks and their properties

The most relevant mobile networks for WebRTC at the moment are Wi-Fi and the different variants of cellular technologies.

Many characteristics of the cellular networks are covered in Section 3 in the context of the particular issue under discussion. The following is a very brief description of the power consumption related properties of WCDMA/HSPA networks. The details vary, but similar principles apply to other cellular networks, at least GPRS/EDGE and LTE.

In simplified terms, the WCDMA/HSPA radio can be in three different types of states: The power-save state (IDLE, Cell\_PCH, URA\_PCH), a shared channel state (Cell\_FACH) or a dedicated channel state (Cell\_DCH). The power-save states consumes about two decades less power than the dedicated channel state, while the shared-channel state is somewhere in the middle. The state machine works so that if

a device has only small packets (upto ~200-500 bytes) to send or receive, it will allocate a shared channel, that operates on low data rate. If there is more traffic (even a single full size IP packet), a dedicated channel is allocated. Starting from the power-save state, the channel allocation typically takes somewhere between 0.5 and 2 seconds, depending on the network and the exact power-save state. Only after that, the first packet is really sent. If two cellular devices were to exchange packets with each other starting from the power-save state, the initial IP-level RTT could be easily 3-4 seconds.

The channel is kept for some time after the last packet has been sent or received. The dedicated channel drops to power-save via the shared channel. The timers from dedicated to shared and shared to power-save are network dependent, but typically somewhere between 5 and 30 seconds. So, in some networks sending a single ping every 30 seconds is enough to keep the power consumption constantly at the maximum level, while in others the power-save state is entered much faster. The total radio power consumption does not actually depend so much on overall volume of traffic, but on how long a dedicated or shared channel is active. So, for instance a 1 kB keep-alive sent every 30 seconds for an hour (total ~100 kB of traffic) consumes much more (even an order or magnitude more!) than a single 10 MB download, assuming that will finish in a minute or two.

The applications have no control over the radio states, but the Operating System and the Radio Modem software can do something about them. In the newer specifications (and devices and networks) it is possible for the device to explicitly ask the radio channel to be abandoned even immediately after the last packet. For instance, if the device were somehow to know that no new packets are to be sent for some time, it could do such signaling and save power.

The bottom line is that applications and protocols should keep as long intervals between traffic as possible, giving the radio as much low-power time as possible. The intervals that are more than a few seconds may help, but at least intervals that are longer than 30 seconds will definitely help. On the other hand, the initial RTT after an interval will be long. This issue is covered in Sections 3.1 and 3.2.

The other key characteristic of cellular networks is that they have long buffers and run link-layer in "acknowledged" mode, meaning all lost packets are retransmitted. This means TCP will easily create long delays and ruins real-time traffic. This is covered in Section 3.4.

The third characteristic is that mobile devices often change networks

on the fly, typically between cellular and Wi-Fi. Most devices only run a single interface at a time. From networking perspective this means that the device's IP address changes, and e.g. all its TCP connections are lost. This is covered in Section 3.3.

### 3. Specific issues and how to deal with them

#### 3.1. Persistent connectivity to the Calling Site

Many WebRTC apps want to be reachable for incoming sessions (JSEP Offers) or other types of asynchronous messages. For this purpose they need some kind of a persistent communication channel with their "Calling Site". Two standard approaches for this are WebSockets and HTTP long-polling. In both of these cases a TCP connection is used as the underlying transport.

Most cellular networks have a firewall preventing incoming TCP connections, even when they allocate public IPv4 or IPv6 addresses. Also NATs are becoming more popular with the exhaustion of IPv4 address space. The firewall and NAT timers for TCP can range between 1 and 60 minutes, depending on the network. To keep the TCP connection alive, the application needs to send some kind of a keep-alive packets with high enough frequency to avoid the timeout.

If the WebRTC app intends to run for a long periods of time (even when the user is not actively interacting with it), it is of utmost importance to keep this keep-alive traffic as infrequent as possible. Every wake-up of the radio consumes a significant amount of power, even if it is needed just for sending and receiving a couple of IP packets. It makes a huge difference, if there are for instance 6 vs. 60 of these wake-ups every hour. A naive application may want to make it sure it sends frequently enough for all possible networks. That leads to unacceptable power consumption. A smarter application will try to figure out a suitable timeout for a given network it is using, and can save a lot of power in networks with longer timers.

There are further strategies to manage the keep-alives so that they consume least amount of power. It is best to send as small keep-alive messages as possible. HSPA/WCDMA networks have a special shared radio channel (FACH) that can carry small amounts of traffic. Its power consumption is typically less than half of the dedicated channel. Depending on the network, a packet of a couple of hundred bytes will usually only require FACH, while a thousand byte packet will require the dedicated channel to be activated. So, a WebSocket PING-PONG is better than an HTTP POST or GET with all the Cookies and other headers attached. If there are multiple applications or connections to be kept alive, the Browser or the underlying platform

should offer some kind of a synchronization for them, so that the radio is woken only once per cycle.

The most efficient approach would be to multiplex the initial incoming messages for all applications over the same TCP connection. This would require the use of some kind of a gateway service in the network. Such "notification" services are available on many platforms, but at the moment they are not typically available for browsers or web applications. It would be useful to standardize or develop Javascript APIs for this purpose. There is W3C work on Server-sent events. Also, the Open Mobile Alliance (OMA) has started work on standardized "notification" services. Be the services standards based or proprietary, the most relevant part to get done would be to give WebRTC and other Web applications access to them. Such services are always subject to privacy concerns, so at minimum the messages passed over them should be end-to-end encrypted. (Traffic analysis threats would still remain.)

### 3.2. Media and Data channels

Real-time media (audio, video) is typically sent and/or received constantly, while the media channel is established. This means radio needs to be on constantly, and there is little for the application to do to preserve power. (Choosing a hardware accelerated video codec over a non-HW-supported one is one thing the application may be able to influence.) At least in LTE there are techniques called Discontinuous Transmission/Reception (DTX, DRX), that operate even in the timeframe of tens of milliseconds and can affect power consumption e.g. for VoIP. It is an open issue if WebRTC stacks can be somehow optimized for them.

The Data Channel may however be often low-volume or even idle for long periods of time. For instance an IM connection may be idle for minutes or even hours. There can be many apps that want to keep such a connection available just in case there is some traffic to be sent or received infrequently. The WebRTC Data Channel is based on SCTP over DTLS over UDP. This means it needs keepalives in the order of 30 seconds in cellular networks, meaning the radio will be active most of the time even if no user traffic is sent. It is not possible to keep such a channel on for a long time due to power consumption.

Applications can choose different strategies to deal with this problem. One approach is to avoid Data Channels completely for low-volume or infrequent traffic and send it via the Web servers over HTTP or WebSockets. This is probably the best approach. The other approach is to tear down the Data Channel after some timeout and re-establish it only when new traffic needs to be sent. This may create some lag in sending the first message after the interval. The third

option is to transport the Data Channel over TCP, e.g. using a yet undefined "HTTP tunneling fallback" mechanism. This would be almost identical to the first approach, except that logically the application would still be using a WebRTC Data Channel. It is not yet clear if this will be feasible due to ICE concent refreshes that may need to occur frequently as well (every 30 seconds?). They are sent end-to-end so one side of the Data Channel can not by itself even affect their rate.

### 3.3. Recovery from interface switching

Most mobile platforms only support Internet connectivity over only one interface at a time. In practice this is either a cellular or a Wi-Fi interface. From radio hardware perspective there would be no need for such a limitation, but it is driven by simplicity and power preservation. The devices typically have a hard-coded or configurable priority order for different networks. The most common policy is that any known Wi-Fi network is always preferred over any cellular network, but even more complex policies are possible.

When the device detects a higher priority network than the one currently in use, it will by default attach to that network automatically. After a successful attachment to the new network, the device turns the old network (and interface) off. In most platforms applications have no control over this. In a typical situation the switch-over leads to a change of IP address, and for instance all TCP connections becoming disconnected, and any state tied to them needs to be recreated.

It is important that WebRTC applications are made robust enough to survive this behavior. Many native applications deal with it by listening to "disconnect" and "reconnect" events through the APIs they are using. For WebRTC apps the first priority is to re-establish its "signaling" connectivity to the "Calling Site". If that connectivity is based on a WebSocket, the application needs to react to the "onerror" event through the WebSocket API and establish a new connection and setup all state related to it. (Say, if the application was using SIP over WebSockets, it might have to re-REGISTER on the SIP level.) If the disconnect was caused by interface switching and the switch-over succeeded cleanly, it would be possible to setup the new connection immediately. In some cases the disconnect could last longer, and the application would have to retry the connection until connectivity is regained.

It would be advisable to make the reconnect step as lightweight as possible in terms of RTTs required. For the browser and the web application platform, it is important that the "disconnect" event gets propagated to the applications as fast as possible.

For HTTP long-polling, it would similarly be important to notice that the underlying TCP connection has become stale, and a new poll needs to be sent as quickly as possible.

The application may also attempt to update any peer-to-peer sessions it is having at the time of the switch-over. At this point of RTCWeb standardization it is not yet clear how much control over this the protocols and APIs will exhibit. There are many layers on which the recovery can be done. It is possible to try to deal with it using ICE. This would require knowing when the currently used ICE candidate becomes unusable, as it is bound to a removed interface. The failure of ICE connectivity checks provide that information, but possibly after some delay. (Frequent connectivity checks are not an issue as long as media is actively sent or received, but would be costly over an idle or low-volume media channel, such as a Data Channel. If media traffic is infrequent, the speed of detection may not be that critical for user experience anyway.) If an interface really became unusable, it would be better to have an explicit event to signal that all ICE candidates bound to it are likely unusable as well, so the application could act immediately. If a new interface became available, the application could restart ICE and start using the new candidates gathered.

The PeerConnection API offers a few events for these purposes, at least "icechange" and "renegotiationneeded". With these the application can learn about problems with the currently used candidates. There is also a method "updateIce" by which the application can restart the ICE candidate gathering process. It is however not yet entirely clear how these event handlers and methods should be best used to deal with an interface change, and whether they even are a feasible tool for dealing with it. It is also important to note that no new offers or answers could be sent or received until the "signaling channel" (e.g. the Websocket connection) was first re-established.

If the lower-level instruments fail, the application could create a new PeerConnection, and recreate the media channels. This would be a heavier operation, but in some cases it might still be better than leaving the recovery entirely to the user, i.e. explicitly making a new call from the UI.

There are certain things that the underlying platform (Operating System, Connection Manager etc.) can also implement to make interface switching smoother for the applications. One possibility would be to keep the old interface available for a short duration even after a new higher priority interface becomes available. This would allow applications to deal with the change in a more proactive fashion. There are also protocols such as Multipath TCP that could be used to

switch e.g. WebSocket connections to a new interface without always resorting to the application support.

### 3.4. Congestion avoidance

Cellular mobile networks have notoriously large buffers. Their link layers also typically operate in an "acknowledged" mode, meaning that the lost frames (or packets) are retransmitted. Retransmission creates head of line blocking on the queue. This means packets are seldom lost, but delays grow large. The individual users or endpoints are often isolated from each other so that the network capacity is divided among them more or less evenly. However, all traffic to and from the same endpoint ends up in the same queue. In WebRTC context this means that plain TCP traffic will easily ruin real-time traffic due to the buffering.

WebRTC protocols should be desinged to avoid this. If Data Channels transfer a lot of data in parallel to the real-time streams, they should not use the loss-driven (TCP) congestion control algorithms but something that reacts to queue growth much faster. IETF LEDBAT WG may have something to offer for this case. If the browser wants to protect its real-time strams in general against all TCP (HTTP, WebSocket) traffic, it might be best for it to also restrict the number of simultaneous TCP connections in use, for instace to retrieve a website. The HTTP 2.0 work done in IETF HTTPBIS WG should prove helpful in this case.

Cellular networks also do have their in-built Quality of Service mechanisms that can be used to differentiate service for different packet flows. These are not widely used in HSPA/WCDMA, but LTE may change the situation to some extent. The QoS policy is enforced by the network, and requires a contract with the operator. It is thus likely only available for services with some relation to the access operator. How the WebRTC application or the browser deal with that is TBD. Technically DiffServ marking is probably the only dynamic approach to indicate the priority of a particular flow.

## 4. Security Considerations

Not explicitly covered in this version.

## 5. Acknowledgements

Bernard Aboba and Goeran Eriksson provided useful comments to the document. Dan Druta has worked on Web notifications in the context of WebRTC.

## 6. References

### Author's Address

Markus Isomaki  
Nokia  
Keilalahdentie 2-4  
FI-02150 Espoo  
Finland

Email: markus.isomaki@nokia.com



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 10, 2013

S. Dhesikan  
Cisco  
D. Druta  
ATT  
C. Jennings  
P. Jones  
J. Polk  
Cisco  
July 9, 2012

DSCP and other packet markings for RTCWeb QoS  
draft-jennings-rtcweb-qos-00

Abstract

Many networks, such as Service Provider and Enterprise networks, can provide per packet treatments based on Differentiated Services Code Points (DSCP) on a per hop basis. This document defines the recommended DSCP values for browsers to use for various classes of traffic.

This draft is a very early and far from done. It is meant to provide the structure for the idea of how to do this but much discussion is needed about the details.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 10, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the

document authors. All rights reserved.

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

## 1. Introduction

DiffServ style packet marking can help provide QoS in some environments. There are many use cases where such marking does not help, but it seldom make things worse, if packets are marked appropriately. In other words, when attempting to avoid congestion by marking certain traffic flows, say all audio or all audio and video, marking too many audio and/or video flows for a given network's capacity can prevent desirable results. Either too much other traffic will be starved, or there is not enough capacity for the preferentially marked packets (i.e., audio and/or video).

This draft proposes how browser and other VoIP applications can mark packets. This draft does not contradict or redefine any advice from previous IETF RFCs but simply provides a simple set of recommendations for implementors based on the previous RFCs.

There are some environments where priority markings frequently help. These include:

1. If the congested link is the broadband uplink in a Cable or DSL scenario, often residential routers/NAT support preferential treatment based on DSCP.
2. If the congested link is a local WiFi network, marking may help.
3. In some cellular style deployments, markings may help in cases where the network does not remove them.

Traditionally DSCP values have been thought of as being site specific, with each site selecting its own code points for each QoS level. However in the RTCWeb use cases, the browsers need to set them to something when there is no site specific information. This document describes a reasonable default set of DSCP code point values drawn from existing RFCs and common usage. These code points are

solely defaults. Future drafts may define mechanisms for site specific mappings to override the values provided in this draft.

This draft defines some inputs that the browser can look at to determine how to set the various packet markings and defines the a mapping from abstract QoS policies (media type, priority level) to those packet markings.

## 2. Terminology

TODO - add the boiler plate

## 3. Inputs

The first input is the type of the media. The browser provides this input as it knows if the media is audio, video, or data. In this specification, both interactive and streaming media is included. They are treated in different categories as their QoS requirements are slightly different. The second input is the relative treatment of the stream within that session. Many applications have multiple video streams and often some are more important than others. Javascript applications can tell the browser whether a particular media stream is high, medium, or low importance to the application.

## 4. DSCP Mappings

	Low	Medium	High
Audio	46 (EF)	46 (EF)	46 (EF)
Interactive Video	38 (AF43)	36 (AF42)	34 (AF41)
Non-Interactive Video	26 (AF33)	28 (AF32)	30 (AF31)
Data	8 (CS1)	0 (BE)	10 (AF11)

Table 1

## 5. QCI Mapping

	Low	Medium	High
Audio	1	1	1
Interactive Video	2	2	2
Non-Interactive Video	8	6	4
Data	9	9	3

Table 2

This corresponds to the mapping provided in TODO REF which are: QCI values (LTE)

Value			Use
1	GBR	2	Interactive Voice
2	GBR	4	Interactive Video
3	GBR	5	Non-Interactive Video
4	GBR	3	Real Time Gaming
5	Non-BG	R 1	IMS Signalling
6	Non-BG	R 7	interactive Voice, video, games
7-9	Non-BG	R 6	non interactive video / TCP web, email, / Platinum vs gold user

Table 3

## 6. WiFi Mapping

	Low	Medium	High
Audio	6	6	6
Interactive Video	5	5	5
Non-Interactive Video	4	4	4
Data	1	0	3

Table 4

This corresponds to the mappings from TODO REF of

Value		Traffic Type	Access Category (AC)	Designation
1	BK	Background	AC_BK	Background
2	-	(spare)	AC_BK	Background
0	BE	Best Effort	AC_BE	Best Effort
3	EE	Excellent Effort	AC_BE	Best Effort
4	CL	Controlled Loat	AC_VI	Video
5	VI	Video	AC_VI	Video
6	VO	Voice	AC_VO	Voice
7	NC	Network Control	AC_VO	Voice

Table 5

## 7. W3C API Implications

To work with this proposal, the W3C specification would need to provide a way to specify the importance of media and data streams.

The W3C API should also provide a way for the application to find out the source and destination IP and ports of any flow as well as the DSCP value or other markings in use for that flow. The JS application can then communicate this to a web service that may install a particular policy for that flow.

## 8. Security Considerations

TODO - discuss implications of what browser can set and what JS can set

## 9. IANA Considerations

This specification does not require any actions from IANA.

## 10. Acknowledgements

Thanks for hints on code to do this from Paolo Severini, Jim Hasselbrook, Joe Marcus, and Erik Nordmark.

## 11. Appendix: Code Hints

On windows setting the source interface works but BSD, OSX, Linux use weak end-system model and will route out different interface if that looks like a better route. (TODO - Can someone verify this with specific versions?)

In windows you might be able to tell something about priority of an interface for ICE purposes with WlanQueryInterface or GetIfTable.

The specific mechanisms required to set DSCP code points depend on the application platform.

In windows, setting the DSCP is not easy. See Knowledge Base Article KB248611. TODO - add more information about what can be done for windows.

For most unix variants, the following program can set DSCP.

TODO - make this work in V6. For v6 have a look at IPv6\_TCLASS or better the tclass part of sin6\_flowid for IPv6

TODO - Can someone test and report back results of program in iOS, Android, Linux, OSX, BSD.

Example test program:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

#define MSG "Hello, World!"

int
main(void) {
    int sock = -1;
    struct sockaddr *local_addr = NULL;
    struct sockaddr_in sockin, host;
    int tos = 0x60; /* CS3 */
```

```
socklen_t socksiz = 0;
char *buffer = NULL;

sock = socket(AF_INET, SOCK_DGRAM, 0);
if (sock < 0) {
    fprintf(stderr, "Error: %s\n", strerror(errno));
    exit(-1);
}

memset(&sockin, 0, sizeof(sockin));
sockin.sin_family = PF_INET;
sockin.sin_addr.s_addr = inet_addr("11.1.1.1");
socksiz = sizeof(sockin);

local_addr = (struct sockaddr *) &sockin;

/* Set ToS/DSCP */
if (setsockopt(sock, IPPROTO_IP, IP_TOS, &tos,
               sizeof(tos)) < 0) {
    fprintf(stderr, "Error setting TOS: %s\n", strerror(errno));
}

/* Bind to a specific local address */
if (bind(sock, local_addr, socksiz) < 0) {
    fprintf(stderr, "Error binding to socket: %s\n", strerror(errno));
    close(sock); sock=-1;
    exit(-1);
}

buffer = (char *) malloc(strlen(MSG) + 1);
if (buffer == NULL) {
    fprintf(stderr, "Error allocating memory: %s\n", strerror(errno));
    close(sock); sock=-1;
    exit(-1);
}
strcpy(buffer, MSG, strlen(MSG) + 1);
memset(&host, 0, sizeof(host));
host.sin_family = PF_INET;
host.sin_addr.s_addr = inet_addr("10.1.1.1");
host.sin_port = htons(12345);

if (sendto(sock, buffer, strlen(buffer), 0,
           (struct sockaddr *) &host, sizeof(host)) < 0) {
    fprintf(stderr, "Error sending message: %s\n", strerror(errno));
    close(sock); sock=-1;
    free(buffer); buffer=NULL;
    exit(-1);
}
```

```
    free(buffer); buffer=NULL;
    close(sock); sock=-1;

    return 0;
}
```

## 12. Normative References

- [1] Babiarz, J., Chan, K., and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, August 2006.
- [2] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec, J., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, March 2002.
- [3] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.
- [4] Heinanen, J., Baker, F., Weiss, W., and J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, June 1999.
- [5] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

## Authors' Addresses

Subha Dhesikan  
Cisco

Email: [sdhesika@cisco.com](mailto:sdhesika@cisco.com)

Dan Druta  
ATT

Email: [dd5826@att.com](mailto:dd5826@att.com)

Cullen Jennings  
Cisco

Email: [fluffy@cisco.com](mailto:fluffy@cisco.com)

Paul Jones  
Cisco

Email: [paulej@packetizer.com](mailto:paulej@packetizer.com)

James Polk  
Cisco

Email: [jmpolk@cisco.com](mailto:jmpolk@cisco.com)



Behave  
Internet-Draft  
Intended status: Standards Track  
Expires: January 16, 2014

Muthu. Perumal  
D. Wing  
R. Ravindranath  
T. Reddy  
Cisco Systems  
July 15, 2013

STUN Usage for Consent Freshness  
draft-muthu-behave-consent-freshness-04

Abstract

Verification of peer consent before sending traffic is necessary in WebRTC deployments to ensure that a malicious JavaScript cannot use the browser as a platform for launching attacks. A related problem is session liveness. WebRTC application may want to detect connection failure and take appropriate action.

This document describes how a WebRTC browser can verify peer consent to continue sending traffic and detect connection failure.

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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 16, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	2
3. Design Considerations . . . . .	3
4. Solution Overview . . . . .	4
5. W3C API Implications . . . . .	5
6. Interaction with Keepalives used for Refreshing NAT Bindings	5
7. Security Considerations . . . . .	5
8. IANA Considerations . . . . .	6
9. Acknowledgement . . . . .	6
10. Normative References . . . . .	6
Authors' Addresses . . . . .	6

## 1. Introduction

To prevent attacks on WebRTC peers, WebRTC browsers have to ensure the remote peer wants to receive traffic. This is performed both when the session is first established to the remote peer (using ICE connectivity checks), and periodically when for the duration of the session (using the procedure defined in this document).

When a session is first established, WebRTC implementations are required to perform STUN connectivity checks as part of ICE [RFC5245]. That initial consent is not described further in this document.

Related to consent is loss of connectivity ("liveness"). WebRTC applications want notification of connection failure to take appropriate actions (e.g., alert the user, try switching to a different interface).

This document describes a new STUN usage with a request and response which verifies the remote peer consents to receive traffic, and detects loss of liveness. To meet the security needs of consent, the JavaScript application has no control over the consent requests or the requirement to receive a consent response. However, the JavaScript does get notification of consent failure and loss of connectivity.

## 2. Terminology

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

**Consent:** It is the mechanism of obtaining permission from the peer to send traffic on a candidate pair.

**Consent Freshness:** It is the mechanism of obtaining permission from the peer to continue sending traffic on a nominated candidate pair after ICE has concluded.

**Session Liveness:** It is the mechanism of detecting connectivity on a nominated candidate pair after ICE has concluded.

**Transport Address:** The combination of an IP address and port number (such as a UDP or TCP port number).

### 3. Design Considerations

Although ICE requires periodic keepalive traffic to be sent in order to keep NAT bindings alive (Section 10 of [RFC5245], [RFC6263]), those keepalives are send-and-forget, and do not evoke a response. A response is necessary both for consent to continue sending traffic, as well as to ensure connectivity. Thus, we need a request/response mechanism.

Though ICE specifies STUN Binding indications to be used for keepalives, it requires that an agent be prepared to receive connectivity check as well. If a connectivity check is received, a response is generated, but there is no impact on ICE processing, as described in section 10 of [RFC5245].

While a WebRTC browser could verify whether the peer continues to send SRTCP reports before sending traffic to the peer, the usage of SRTCP together with Security Descriptions [RFC4568] requires exposing the media keys to the JavaScript and renders SRTCP unsuitable for consent freshness.

For consent, calculating the SHA1 HMAC is necessary for MESSAGE-INTEGRITY which is computationally expensive. Security analysis concluded that the STUN 96 bits transaction ID is sufficient for consent, without needing MESSAGE-INTEGRITY. However, omitting the MESSAGE-INTEGRITY attribute from STUN Binding request/response to avoid the cost of computing SHA1 would make browsers incapable of verifying consent freshness with legacy ICE/ICE-lite implementations, resulting in backward compatibility issues.

The above considerations suggest that STUN Binding request/response is most suitable for performing consent freshness.

#### 4. Solution Overview

Consent freshness serves as a circuit breaker (so that if the path or remote peer fails the WebRTC browser stops sending all traffic on that remote peer), determining session liveness serves the purpose of notifying the application of connectivity failure so that the application can take appropriate action.

The solution uses three values:

1. A consent timer,  $T_c$ , whose value is determined by the browser. This value **MUST** be 15 seconds.
2. A packet receipt timer,  $T_r$ , whose value is determined by the application, but **MUST NOT** be shorter than 1 second or longer than 15 seconds, and **SHOULD** have a default value of 5 seconds.
3. A consent timeout,  $T_f$ , which is how many seconds elapse without a consent response before the browser ceases transmission of media. Its value **MUST** be 15 seconds or less, and the value 15 seconds is **RECOMMENDED**.

A WebRTC browser performs a combined consent freshness and session liveness test using STUN request/response as described below:

Every  $T_c$  seconds, the WebRTC browser sends a STUN Binding Request to the peer. This request **MUST** use a new, cryptographically random Transaction ID [RFC4086], and is formatted as for an ICE connectivity check [RFC5245]. A valid STUN Binding Response is also formatted as for an ICE connectivity check [RFC5245]. The STUN Binding Request and STUN Binding Response are validated as for an ICE connectivity check [RFC5245].

If a valid STUN Binding Response is received, the consent timer is reset and fires again  $T_c$  seconds later.

If a valid STUN Binding Response is not received after 500ms, the STUN Binding Request is retransmitted (with the same Transaction ID and all other fields). As long as a valid STUN Binding Response is not received, this retransmission is repeated every 500ms until  $T_f$  seconds have elapsed or a valid response is received. If no valid response is received after  $T_f$  seconds, the WebRTC browser **MUST** quit transmitting traffic to this remote peer. Considering the default value of  $T_f=15$  seconds, this means transmission will stop after 30 consent check packets have resulted in no response.

Liveness timer: If no packets have been received on the local port in  $T_r$  seconds, the WebRTC browser MUST inform the JavaScript that connectivity has been lost. The JavaScript application will use this notification however it desires (e.g., cease transmitting to the remote peer, provide a notification to the user, etc.). Note the definition of a received packet is liberal, and includes an SRTP packet that fails authentication, a STUN Binding Request with an invalid USERNAME or PASSWORD, or any other packet.

## 5. W3C API Implications

For the consent freshness and liveness test the W3C specification should provide APIs as described below:

1. Ability for the browser to notify the JavaScript that a consent freshness transaction has failed for a media stream and the browser has stopped transmitting for that stream.
2. Ability for the JavaScript to start and stop liveness test and set the liveness test interval.
3. Ability for the browser to notify the JavaScript that a liveness test has failed for a media stream.

## 6. Interaction with Keepalives used for Refreshing NAT Bindings

When not actively sending traffic on a nominated candidate pair, performing consent freshness does not serve any purpose from a security perspective. If consent freshness is not performed during this period, the browser continues to perform the ICE keepalives [RFC5245] or RTP keepalive [RFC6263] to refresh NAT bindings.

## 7. Security Considerations

Security considerations discussed in [RFC5245] are to be taken into account.

In ICE [RFC5245] the STUN request/response are protected with MESSAGE-INTEGRITY, using an ephemeral username and password exchanged in the SDP ice-ufrag and ice-pwd attributes. This prevents ICE from accidentally connecting to an in-intended peer, in that ICE will only connect to a peer that also knows the same username and password (exchanged in call signaling). Once that connection to the remote peer has been established with ICE, the consent to continue sending traffic does not benefit from re-asserting that same username and password, so long as the senders and receiver's IP addresses remain the same (as they usually do).

## 8. IANA Considerations

This document does not require any action from IANA.

## 9. Acknowledgement

Thanks to Eric Rescorla, Harald Alvestrand, Martin Thomson, Bernard Aboba, Cullen Jennings and Simon Perreault for their valuable inputs and comments.

## 10. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, June 2011.

## Authors' Addresses

Muthu Arul Mozhi Perumal  
Cisco Systems  
Cessna Business Park  
Sarjapur-Marathahalli Outer Ring Road  
Bangalore, Karnataka 560103  
India

Email: mperumal@cisco.com

Dan Wing  
Cisco Systems  
821 Alder Drive  
Milpitas, California 95035  
USA

Email: [dwing@cisco.com](mailto:dwing@cisco.com)

Ram Mohan Ravindranath  
Cisco Systems  
Cessna Business Park  
Sarjapur-Marathahalli Outer Ring Road  
Bangalore, Karnataka 560103  
India

Email: [rmohanr@cisco.com](mailto:rmohanr@cisco.com)

Tirumaleswar Reddy  
Cisco Systems  
Cessna Business Park, Varthur Hobli  
Sarjapur Marathalli Outer Ring Road  
Bangalore, Karnataka 560103  
India

Email: [tiredddy@cisco.com](mailto:tiredddy@cisco.com)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: February 21, 2013

O. Ohlsson  
Ericsson  
August 20, 2012

Support of SDES in WebRTC  
draft-ohlsson-rtcweb-sdes-support-01

Abstract

Which key management protocols to support has been lively debated in WebRTC on several occasions. This document explains the benefits of SDES and argues why allowing it as an alternative option has little impact on security.

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 <http://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 February 21, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	3
2. Benefits of Supporting SDES . . . . .	3
2.1. Reduced Complexity of WebRTC-SIP Gateway . . . . .	3
2.2. Reduced Processing (Less SRTP Terminations) . . . . .	3
2.3. Reduced Call Setup Time . . . . .	4
3. Security Considerations . . . . .	5
3.1. SDES in case of an Outside Attacker . . . . .	5
3.1.1. Extraction of Log Data . . . . .	6
3.1.2. Script Injection . . . . .	6
3.2. SDES in case of an Inside Attacker . . . . .	7
3.2.1. Downgrade Attack . . . . .	7
3.2.2. Difficulties with Key Continuity . . . . .	8
3.2.3. 3rd Party Identity Assertion . . . . .	9
4. Discussion and Conclusion . . . . .	9
5. Informative References . . . . .	10
Author's Address . . . . .	11

## 1. Introduction

Which key management protocols to support has been lively debated in WebRTC on several occasions. The main question is the following: Should applications be restricted to DTLS-SRTP or could SDES be allowed as an alternative option?

In this document we identify and address the issues that have been raised. We explain the benefits of SDES and argue why allowing it as an alternative option has little impact on security.

## 2. Benefits of Supporting SDES

Being able to communicate from WebRTC applications to existing SIP/RTP endpoints is a highly desirable use case. The SIP installed base is huge and contains millions of devices and a large number of applications (e.g. conferencing and voicemail). Even more important, nearly all mobile phones and landlines are reachable through SIP/RTP gateways deployed in service provider networks. The same can also be said for other signaling protocols, such as XMPP or H.323. As a sidenote, the recent work on the DTMF tone API in WebRTC proves that many members consider legacy interworking to be important.

### 2.1. Reduced Complexity of WebRTC-SIP Gateway

Communication between the Browser and SIP/RTP endpoint will most likely require some form of media-plane gateway (due to the need to terminate ICE). The development and testing costs for such gateways are typically very high since they need to handle a large number of users and often contain special purpose hardware. It is definitely worthwhile to try to reduce costs by lowering the complexity and removing functionality that is not strictly required. This would result in lower prices which will lead to a higher degree of interconnectivity between WebRTC and existing SIP deployments.

Already today there are Session Border Controllers (SBC) that perform SRTP termination on behalf of endpoints with SDES based keying (there are SBCs that support DTLS-SRTP but this is uncommon). If the browser also supported SDES, the WebRTC gateway could simply forward all SRTP packets to the SBC and let it decide whether to terminate encryption or not (depending on the capabilities of the receiving endpoint).

### 2.2. Reduced Processing (Less SRTP Terminations)

A large part of modern SIP/RTP devices support SRTP and most of them that do, use SDES based keying. This is confirmed in the report from

the latest [SIPit] event which stated that:

- o 80 percent of the tested implementations supported SRTP
- o 100 percent of the SRTP implementations supported SDES
- o 0 percent of the SRTP implementations supported DTLS-SRTP

Although these figures may not be entirely accurate, they at least provide an indication of the current situation.

The 3rd Generation Partnership Project (3GPP) has also selected SDES for key management in the IP Multimedia Subsystem (IMS) [3GPP.33.328]. We can therefore expect the number of SDES capable devices to increase as Voice over LTE (VoLTE) and other IMS based systems become more widely deployed.

Provided SDES is included in browsers, calls between the WebRTC and SIP domains do not need to be encrypted/decrypted by an intermediate gateway when the SIP endpoint supports SDES. This leads to a substantial reduction in processing cost for the gateway in SIP domains where a large part of the devices support SDES. Another benefit is that for those endpoints that support SDES the call will be protected end-to-end for free. Achieving this with DTLS-SRTP would require the gateway to first decrypt and then re-encrypt traffic.

Note that the important question is whether the gateway needs to terminate SRTP at all. Processing wise there is probably not that much difference in terminating an SRTP + SDES or an SRTP + DTLS-SRTP call.

DTLS-SRTP with Encrypted Key Transport (EKT) [I-D.ietf-avtcore-srtp-ekt] has been suggested as an alternative to avoid expensive encryption/decryption in gateways. If browsers support DTLS-SRTP with EKT, a gateway can force the browser and the SDES endpoint to agree on the same set of SRTP keys and algorithm settings. Once this is done, the gateway will simply forward the SRTP (and SRTCP) packets in both directions. The downside of using this approach is the increased complexity of the gateway (new protocols and additional signaling are required) and the lack of implementation experience.

### 2.3. Reduced Call Setup Time

With SDES a peer can begin to send media as soon as an ICE candidate pair has been nominated for use and the connectivity check for that pair has succeeded. If DTLS-SRTP is being used the peer would also

need to wait for the DTLS-SRTP handshake to complete, which requires two additional roundtrips.

Obviously, being able to start sending media quickly is not very useful unless the receiver knows how to process the incoming packets. One common argument against SDES is its inability to handle early media (i.e. media that arrives at the SDP offerer before the SDP answer arrives). However, this problem cannot occur if the offerer is ICE full. To see why, recall that sending media requires that a candidate pair has been nominated for use by the ICE controlling agent, which is always the offerer when the offerer is ICE full. Since nomination is done by sending a connectivity check (with the nomination flag set) which requires the password provided in the SDP answer, no pair gets nominated at the answerer and no media is sent before the SDP answer has arrived at the offerer.

If the offerer is ICE lite or if multiplexing is used (i.e. all media streams are sent over a single ICE candidate pair) and an additional media stream is added later in time via an updated offer, then the problem with early media could arise when SDES is used (but never with DTLS-SRTP).

### 3. Security Considerations

At this point most readers should agree that SDES is favourable from an interworking point of view. It is also clear that implementing SDES in WebRTC is a relatively straight forward task. What remains to be considered are its impacts on security.

We distinguish between the following two types of attackers:

Outside Attacker	An external party attempts to intercept a call (e.g. a host located on the same WLAN as the user)
Inside Attacker	The web application itself (or the signaling server, in case the web server and signaling server are separated) attempts to intercept a call

#### 3.1. SDES in case of an Outside Attacker

By requiring that signaling is secured using TLS, an outside attacker that monitors network traffic will not be able to extract the SDES keys. Therefore, in this scenario both SDES and DTLS-SRTP provide a sufficient level of protection.

The two other types of attacks that have been mentioned in this context are extraction of log data and code injection, each of which are considered below.

#### 3.1.1. Extraction of Log Data

In this scenario the attacker manages to decrypt a previously recorded call by attacking the signaling server and extracting the SDES keys from the server log.

First of all, if the attacker gets as far as reading the logging data then eavesdropping of past calls is probably not the only problem. The effort required to break into the server is also related to the amount of trust the user assigns to the web application: well trusted sites often have well protected servers.

Secondly, it can be questioned how common this type of extensive logging really is. Storing passwords and other sensitive information in log files is an implementation mistake that can easily be avoided.

Finally, SDES will primarily be used when interworking with existing SIP systems deployed within enterprises or service providers. These have been using SDES for a long time and know that it is critical to protect the plain text keys.

#### 3.1.2. Script Injection

In this scenario the attacker manages to inject his own piece of JavaScript into the WebRTC application. The next time a user downloads the application and places a call, the script will execute and start eavesdropping on the conversation.

There are three major ways in which code can be injected into a web application:

- o The page itself or one of its included JavaScript files is downloaded over a non-HTTPS link and is modified en route
- o The web application intentionally includes JavaScript supplied by the attacker (e.g. a third-party library or advertisement)
- o HTML form input or URL parameters are not properly sanitized (i.e. classical XSS vulnerability)

Modification en route is prevented by requiring HTTPS to be used for all content. Whether the two other injection techniques are feasible or not largely depends on the application.

If script injection occurs then there are other methods to intercept a call, like establishing additional PeerConnection objects or use a recording interface and send the data using WebSocket. As long as these methods are available it does not matter much whether the application uses SDES or DTLS-SRTP.

In general, if an attacker manages to execute even a small piece of JavaScript then he has effectively gained full control of the application (additional code can be included and HTML elements removed/inserted). Since this situation is exactly the same as the situation with an inside attacker, script injection will not be discussed further.

### 3.2. SDES in case of an Inside Attacker

First of all, it can be questioned if we really want to protect ourselves against an inside attacker. If consent is required every time the application wants to record or forward media then the user experience will suffer. One could also imagine future applications that want to use their own codecs or filters (for example a voice scrambler or face detection software), something which is difficult to achieve without access to the underlying bitstreams.

We ignore this problem for now and simply assume that the application cannot access the media from within the browser. In other words, we only consider protection of the media during transport.

#### 3.2.1. Downgrade Attack

The major argument against SDES is that it would make it trivial for the application to perform interception. Let us compare what would be required in both cases.

Interception of SDES call:

1. Copy and store the 'a=crypto:' lines in the offer/answer SDP
2. Force media to pass through TURN server by deleting all candidates except the relayed one
3. Store all SRTP packets that pass through the TURN server and decrypt them later on (using the keys from step 1)

Interception of DTLS-SRTP call:

1. Replace the 'a=fingerprint:' lines in the offer/answer SDP with the fingerprint of a public key generated by the application

2. Force the media to go through the TURN server by deleting all ICE candidates except the relayed one
3. Modify an existing TURN server implementation so that it decrypts and re-encrypts the DTLS traffic (using the public-private key pair from step 1)

Putting the modified TURN server into place is the hardest part of intercepting a DTLS-SRTP call. Once this is done however, the remaining steps are fairly straightforward. This shows that neither DTLS-SRTP nor SDES provides any significant protection against an inside attacker.

There is one benefit of DTLS-SRTP that is not directly apparent from the above description. If both users read their respective fingerprint values over the voice channel then they can detect if the conversation is being intercepted. However, it is very unlikely that the average user would bother doing this.

### 3.2.2. Difficulties with Key Continuity

The comparison in the previous section is somewhat simplified since it does not consider DTLS-SRTP key continuity. The way this mechanism works is that the browser will notify the user whenever it receives a certificate which has not previously been seen (i.e. not present in the browser cache). Since the user will receive this notification every time he calls someone new and whenever someone changes browser, it is very likely that he/she will simply ignore it.

Reuse of public keys also has privacy implications as it enables user tracking. A user that wants to remain anonymous towards a service provider would need to generate a fresh key for each interaction. Furthermore, in order to avoid colluding service providers (e.g. medical clinics and insurance agencies) from linking a user's activities, separate certificates are needed for different domains. However, storing domain names together with the certificates might allow the next browser user (e.g. a family member) to see which sites the previous user visited. All of this leads to more certificates being generated which in turn results in even more "new key" notifications.

It is also important to understand that the cached certificates are not bound to any identity (the certificates are simple containers for the public key without any additional information). This means that if just one of the cached keys is compromised any user call can be intercepted without causing the "new key" notification to be displayed. Note that the risk of this happening is directly related to the size of the cache, which grows over time.

### 3.2.3. 3rd Party Identity Assertion

[I-D.rescorla-rtcweb-generic-idp] suggests a way to strengthen the security of DTLS-SRTP by validating the received fingerprint via an identity provider. At the time of writing there are still some details missing from the proposal (for example, it is not clear how the identity provider is selected in practice or how the peer identity is displayed to the user) but it definitely seems promising. Such a mechanism (including the necessary browser chrome) would make it significantly harder for the application to act as man-in-the-middle.

The question is whether the identity mechanism is optional or not, i.e. will it be possible for an application to use "plain" DTLS-SRTP. The answer is most likely "yes" due to the following reasons:

- o Many applications are already trusted by the user
- o Some applications do not want to depend on third parties
- o Some users do not have any identity provider account
- o Users may not always want to reveal their identity
- o Working out all the details of the identity mechanism will take time (and if it is not mandatory from start there are backward compatibility issues)

Note that allowing an application to be its own identity provider is effectively the same as allowing plain DTLS-SRTP (the user trusts the application) only more complicated.

## 4. Discussion and Conclusion

We are not looking to replace DTLS-SRTP with SDES. The 20-line WebRTC developer will continue to use the default option which is DTLS-SRTP, while others who are interested in interworking will select SDES. The latter group will be required to use HTTPS for all content and can be informed of the necessary precautions (secure storage of log files or otherwise no extensive logging).

The main issue that appears to concern members is the application's ability to downgrade security. But as we have seen it is not significantly harder for the application to attack DTLS-SRTP. The main advantage of DTLS-SRTP is the possibility to detect when a call is being intercepted. However, doing so requires an effort from the user and a certain degree of technical skill.

It has been suggested that additional identity mechanisms could prevent the application from listening in on calls. While this is certainly true, any such mechanism would most likely be made optional. If that is the case or if an application can be its own identity provider, then we are back at the situation where the user has to decide which sites to trust.

It can also be questioned to what extent the application should be restricted from accessing media since this limits usability and innovativity. The W3C would need to update its specifications and ensure that a web application cannot record or forward a `MediaStream` without permission from the user.

## 5. Informative References

[3GPP.33.328]

3GPP, "IP Multimedia Subsystem (IMS) media plane security", 3GPP TS 33.328 9.3.0, December 2010, <<http://www.3gpp.org/ftp/specs/html-info/33328.htm>>.

[I-D.ietf-avtcore-srtp-ekt]

McGrew, D., Wing, D., and K. Fischer, "Encrypted Key Transport for Secure RTP", draft-ietf-avtcore-srtp-ekt-00 (work in progress), July 2012.

[I-D.ietf-rtcweb-overview]

Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", draft-ietf-rtcweb-overview-04 (work in progress), June 2012.

[I-D.ietf-rtcweb-use-cases-and-requirements]

Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-09 (work in progress), June 2012.

[I-D.kaplan-rtcweb-sip-interworking-requirements]

Kaplan, H., "Requirements for Interworking WebRTC with Current SIP Deployments", draft-kaplan-rtcweb-sip-interworking-requirements-02 (work in progress), November 2011.

[I-D.rescorla-rtcweb-generic-idp]

Rescorla, E., "RTCWEB Generic Identity Provider Interface", draft-rescorla-rtcweb-generic-idp-01 (work in progress), March 2012.

- [RFC4568] Andreassen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.
- [SIPit] "SIPit27 Summary",  
<[https://www.sipit.net/SIPit27\\_Summary](https://www.sipit.net/SIPit27_Summary)>.

## Author's Address

Oscar Ohlsson  
Ericsson  
Farogatan 6  
SE-164 80 Kista  
Sweden

Email: [oscar.ohlsson@ericsson.com](mailto:oscar.ohlsson@ericsson.com)



RTCWEB  
Internet-Draft  
Intended status: Standards Track  
Expires: January 9, 2013

T. Reddy  
Muthu A M. Perumal  
R. Ram Mohan  
D. Wing  
Cisco  
July 8, 2012

STUN Extensions for Authenticated Firewall Traversal  
draft-reddy-rtcweb-stun-auth-fw-traversal-00

Abstract

Some networks deploy firewalls configured to block UDP traffic. When SIP user agents or WebRTC endpoints are deployed behind such firewalls, media cannot be sent over UDP across the firewall, but must be sent using TCP (which causes a different user experience) or through a session border controller.

This draft describes an alternate model wherein extensions to ICE connectivity checks can be examined by the firewall to permit outgoing UDP flows across the firewall.

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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 9, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Notational Conventions . . . . .	4
3. Problem Statement . . . . .	4
4. Solution Overview . . . . .	5
4.1. Different Components and the Trust model . . . . .	5
5. Usage and Processing . . . . .	6
5.1. Generating FW-FLOWDATA Attribute . . . . .	6
5.2. Sending FW-FLOWDATA Attribute in Binding Request . . . . .	6
5.3. Firewalls processing FW-FLOWDATA Attribute . . . . .	7
6. STUN Attribute Format . . . . .	9
7. Key Provisioning . . . . .	11
8. Security Considerations . . . . .	11
9. IANA Considerations . . . . .	12
10. Acknowledgements . . . . .	12
11. References . . . . .	12
11.1. Normative References . . . . .	12
11.2. Informative References . . . . .	13
Authors' Addresses . . . . .	13

## 1. Introduction

To protect networks using real-time communications, firewalls or session border controllers are typically deployed.

Firewalls include Application Layer Gateway functionality, which intercepts and analyzes the session signaling traffic such as the Session Initiation Protocol (SIP) traffic and creates dynamic mapping to permit the media traffic. In particular, firewall extracts the media transport addresses and ports from the session description and creates dynamic mapping for media to flow through. This model has the following problems:

1. It does not work if the session signaling is end-to-end encrypted (say, using TLS).
2. It does not work if a non-standard session signaling is used that the firewall does not understand.
3. It does not work if the session signaling and media traverse different firewalls.

When an enterprise deploys WebRTC, the above problems are relevant because:

1. The session signaling between the WebRTC application running in the browser and the web server could be using TLS.
2. WebRTC does not enforce a particular session signaling protocol to be used. So, the firewall may not be able to understand it.
3. This session signaling and the peer-to-peer media may traverse different firewalls.

As a result the firewall may block ICE connectivity checks and media traffic.

Session Border Controllers (SBCs) are active participants with call signaling. Like firewalls, they also create dynamic mappings to permit media traffic. This forces call signaling and media through specific IP addresses, belonging to the SBC or an SBC-controlled media relay device.

This draft has the WebRTC server generate a cryptographic token which is passed to the WebRTC endpoint. The endpoint includes the token in its ICE connectivity checks. The firewall intercepts the ICE connectivity checks containing the token, validates it, and permits the ICE connectivity checks and the subsequent media flow through the

firewall.

## 2. Notational Conventions

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

This note uses terminology defined in [RFC5245].

## 3. Problem Statement

In the below topology, a WebRTC server is deployed in the enterprise Data Center. Alice makes a WebRTC call to Bob. For the two endpoints to successfully establish media sessions, firewalls FW1 and FW2 need to permit the ICE connectivity checks and media traffic. In such scenarios the mechanism described in this draft proposes a new comprehension-optional FW-FLOWDATA STUN attribute to be included in STUN Bind requests sent during ICE connectivity checks so that firewalls will permit media traffic between internal peers. This STUN attribute is created by the trusted WebRTC server and sent to the endpoints to be propagated by the respective ICE agents during ICE connectivity checks.

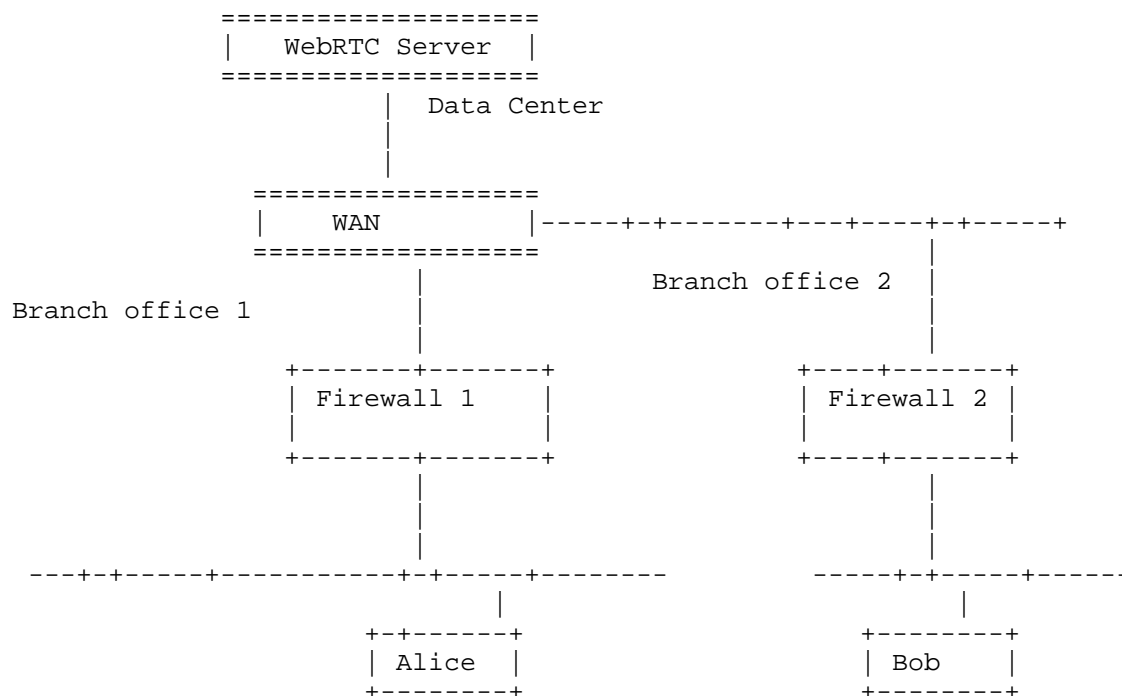


Figure 1: WebRTC service in enterprise - internal call

#### 4. Solution Overview

This section gives an overview of the solution and the different components involved in the solution and the role of each component.

##### 4.1. Different Components and the Trust model

Figure 1 above shows typical components involved in a WebRTC scenario. As part of the call setup, the WebRTC endpoint would have to gather its candidates from a STUN/TURN server, send the candidates in the offer to the peer endpoint. On receiving the answer from the peer endpoint it starts the ICE connectivity checks. As discussed in the problem statement, firewalls would typically block these ICE connectivity checks and media flowing there after. To allow this traffic a firewall needs to authorize the flow.

- o A new comprehensive optional STUN attribute called FW-FLOWDATA is defined as part of this draft. This is used by WebRTC endpoints requiring firewall traversal.

- o This STUN attribute FW-FLOWDATA is generated by the WebRTC server in co-ordination with the WebRTC endpoint.
- o Once the WebRTC session ends the firewall's dynamic mappings are closed after timeout.
- o DISCUSSION: Could we could have a FW-FLOWDATA attribute sent in a STUN message to close the dynamic mappings in the firewalls?

## 5. Usage and Processing

An RTP endpoint which generates media can include the FW-FLOWDATA attribute in its STUN Binding requests used in ICE connectivity checks, to inform on-path firewalls to permit the flow.

### 5.1. Generating FW-FLOWDATA Attribute

The WebRTC server after processing the OFFER/ANSWER sends the FW-FLOWDATA STUN attribute to both the peers to be included in the ICE connectivity checks. The Authentication Tag field in the FW-FLOWDATA attribute contains the digest of the FW-FLOWDATA attribute for data origin authentication and integrity protection. The server first selects the candidate address info based on OFFER/ANSWER exchange and generates other fields of this attribute. The server then computes a digest for the FW-FLOWDATA attribute using HMAC-SHA1. The key for HMAC-SHA1 is provisioned using the technique in Section 7. The result of which is truncated to 96 bits (retaining the left most bits) to produce HMAC-SHA-1-96 and input into the Authentication Tag field. The mechanism to send FW-FLOWDATA attribute from the WebRTC server to the client is outside the scope of this draft. But it is assumed that signalling protocols used for WebRTC call setup will be enhanced to deliver this new attribute to the WebRTC client. The WebRTC server MUST provide a new FW-FLOWDATA to allow the media session to continue before Lifetime expires.

### 5.2. Sending FW-FLOWDATA Attribute in Binding Request

Once a WebRTC endpoint receives the FW-FLOWDATA, it is responsible for generating the STUN message and retransmitting the transactions per the STUN specification. The FW-FLOWDATA attribute should be placed before the FINGERPRINT attribute (if present) and after the MESSAGE-INTEGRITY attribute. The STUN length field is adjusted to point to the new end of the STUN message; that is, the STUN length field always accurately indicates the length of the STUN message (including the MESSAGE-INTEGRITY, FINGERPRINT, and FW-FLOWDATA attributes). This does not interfere with 3rd party receivers of the STUN message, as they will adjust the STUN length field to point to

the end of the MESSAGE-INTEGRITY field. Receivers that do not understand the FW-FLOWDATA will ignore it.

FW-FLOWDATA attribute received by the WebRTC client is passed to the web browser's ICE agent (API to be added in in W3C WebRTC-API specification [I.D.w3c-webrtc]). The ICE agent includes the FW-FLOWDATA attribute with all ICE connectivity checks, so that on-path firewalls can validate and permit the ICE connectivity checks and forthcoming media.

For the FW-FLOWDATA attribute to be visible to the firewalls between the client and the TURN server, the FW-FLOWDATA should be included in the ALLOCATE request, channel bind or refresh messages going to the TURN server. This is to avoid firewalls having to look for STUN packets within STUN (TURN) packets.

### 5.3. Firewalls processing FW-FLOWDATA Attribute

Firewalls can reliably determine a UDP message is a STUN message because all STUN messages sent as ICE connectivity checks include the 32-bit STUN magic cookie and the FINGERPRINT attribute. STUN messages which are authenticated also include a MESSAGE-INTEGRITY attribute which authenticates the fields prior to the MESSAGE-INTEGRITY.

When the firewall receives a STUN binding request with FW-FLOWDATA attribute it stores the Authentication Tag in the FW-FLOWDATA attribute. The firewall then generates a digest for the FW-FLOWDATA attribute using HMAC-SHA1. The result of which is truncated to 96 bits (retaining the left most bits) to produce HMAC-SHA-1-96. If the value of the newly generated digest HMAC-SHA-1-96 is identical to the stored one, the firewall can ensure that the FW-FLOWDATA attribute has not been tampered with. Otherwise the packet is discarded.

To facilitate timestamp checking for replay attacks, each firewall SHOULD store the following information for each host: (The timestamp check is mostly brought from SEND [RFC3971])

- o The receive time of the last received and accepted STUN message with FW-FLOWDATA Attribute. This is called RDlast.
- o The timestamp in the last received and accepted STUN message with FW-FLOWDATA Attribute. This is called TSlast.

When a message is received from a new host (i.e., one that is not stored in the cache), the received timestamp, TSnew, is checked, and the packet is accepted if the timestamp is recent enough to the reception time of the packet, RDnew:

$-\Delta < (RD_{new} - TS_{new}) < +\Delta$

The recommended value for the allowed Delta is 180 seconds. If the timestamp is NOT within the boundaries then discard the STUN message.

When a message is received from a known host (i.e., one that already has an entry in the cache), the timestamp is checked against the previously received STUN message with FW-FLOWDATA Attribute:

$TS_{new} + fuzz > TS_{last} + (RD_{new} - RD_{last}) \times (1 - drift) - fuzz$

If this inequality does not hold, firewall SHOULD discard the message. If, on the other hand, the inequality holds, then firewall SHOULD process the message. Moreover, if the above inequality holds and  $TS_{new} > TS_{last}$ , the receiver SHOULD update  $RD_{last}$  and  $TS_{last}$ . Otherwise, the receiver MUST NOT update  $RD_{last}$  or  $TS_{last}$ . The default value of fuzz is 1 second and drift 1 % (0.01).

The firewall also performs the following checks:

- o Ensures that the source IP address and UDP port of the packet matches with one of the local CAI entries or remote CAI entries in the payload unless the firewall is configured to ignore this check.
- o Ensures the destination IP address and UDP port of the packet matches with one of the remote CAI entries or local CAI entries in the packet payload.

If all the above checks pass then the firewall creates the 5-tuple dynamic mapping using the local candidate IP address, local candidate port, remote candidate IP address, remote candidate port, transport protocol. The session time of the dynamic mapping will be set to a short lifetime (default value of 60 seconds).

o If the initial ICE connectivity check includes the ICE-CONTROLLING attribute but does not include USE-CANDIDATE and a subsequent ICE connectivity check includes both these attributes, the firewall can determine that the ICE agent is the controlling agent using regular nomination and this candidate pair is nominated for media flow. The firewall then sets the session time of the dynamic mapping equal to the Lifetime field in FW-FLOWDATA attribute.

o If the initial ICE connectivity check includes the ICE-CONTROLLED attribute but does not include USE-CANDIDATE and a subsequent ICE connectivity check includes both the attributes, the firewall can determine that the ICE agent is the controlled agent using regular nomination and this candidate pair is nominated for media flow. The

firewall then sets the session time of the dynamic mapping equal to the Lifetime field in FW-FLOWDATA attribute.

- o If the initial ICE connectivity check includes the ICE-CONTROLLING attribute and the USE-CANDIDATE attribute, firewall can determine that the ICE agent is the controlling agent using aggressive nomination. It then waits for the media traffic to flow before setting the session time of the dynamic mapping equal to Lifetime field in FW-FLOWDATA attribute.

- o If the initial ICE connectivity check includes the ICE-CONTROLLED attribute and the USE-CANDIDATE attribute, the firewall can determine that the ICE agent is the controlled agent using aggressive nomination. It then waits for the media traffic to flow before setting session time of the dynamic mapping equal to Lifetime field in FW-FLOWDATA Attribute.

This technique would ensure that dynamic mappings created for pairs in ICE checklist which are not nominated for sending media will be removed after a short duration of time.

DISCUSSION: If WebRTC implementations of RTP support multiplexing of multiple media sessions onto a single RTP session, FW-FLOWDATA attribute can be enhanced to carry a flag indicating the same so that firewall can immediately close the dynamic mapping created for other pairs in the ICE checklist once media starts flowing on one the candidate pairs. In case of multi-homing firewalls can track multiple host IP addresses using authentication supplicant or, for hosts lacking the supplicant, use address-based authentication method.

## 6. STUN Attribute Format

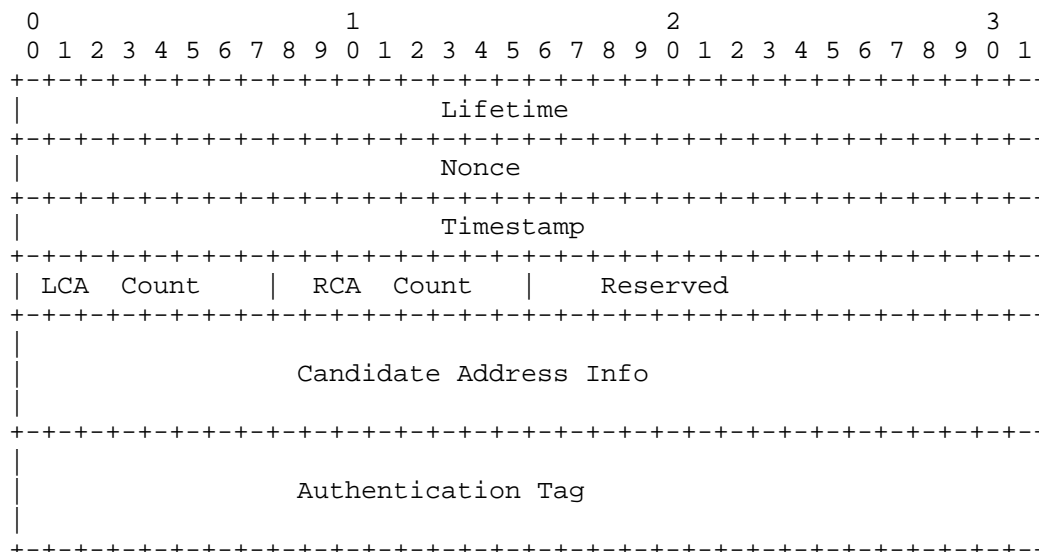


Figure 2: FW-FLOWDATA Attribute

Lifetime: 32-bit unsigned integer. The length of time in seconds that the STUN attribute is valid for the purpose of firewall creating dynamic mapping. The lifetime of the firewall dynamic mapping is set to this value. After the lifetime expires the mapping is deleted, unless the lifetime is extended using a another FW-FLOWDATA attribute.

Nonce: 96-bit unsigned integer. Random value chosen by the WebRTC Server that uniquely identifies the STUN attribute.

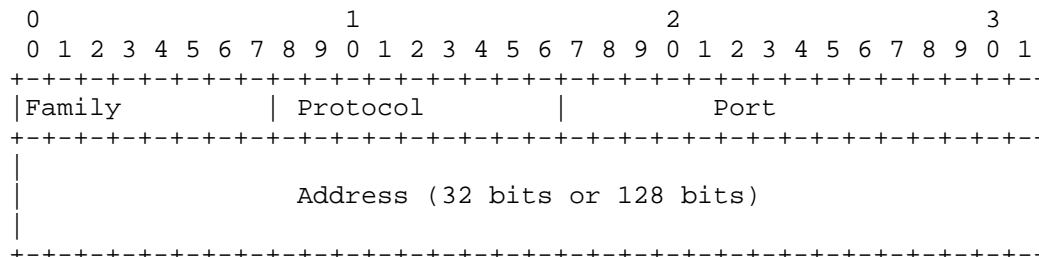
Timestamp: 64-bit unsigned integer field containing a timestamp. The value indicates the number of seconds since January 1, 1970, 00:00 UTC, by using a fixed point format. In this format, the integer number of seconds is contained in the first 48 bits of the field, and the remaining 16 bits indicate the number of 1/64K fractions of a second.

LCA Count: 8-bit unsigned integer. Number of local candidate addresses.

RCA Count: 8-bit unsigned integer. Number of remote candidate addresses.

Reserved: 16-bit unsigned integer. An unused field. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.

Candidate Address Info:



It consists of an 8-bit address family, L4 protocol (for example 17 for UDP, 6 for TCP) and a 16-bit port, followed by a fixed-length value representing the IP address. If the 16-bit port value is 0 it indicates "all ports". The address family can take on the following values: 0x01:IPv4 and 0x02:IPv6.

Authentication Tag: A 96-bit field that carries the Message Authentication Code for the FW-FLOWDATA STUN attribute.

## 7. Key Provisioning

Static keys are preconfigured, either manually or through a network management system. The simplest way to implement FW-FLOWDATA validation is to use static keys. The provisioning of static keys requires either manual operator intervention on the WebRTC Server and each firewall in the enterprise or a network management system performing the same task.

Alternatively using Dynamic Group Key Distribution, group keys are dynamically distributed among the WebRTC server and enterprise firewalls using GDOI [RFC6407]. In this way, each firewall requests a group key from a key server as part of an encrypted and integrity-protected key agreement protocol. Once the key server has authenticated and authorized the firewalls, it distributes a group key to the group member. The authentication in this model can be based on public key mechanisms, thereby avoiding the need for static key provisioning.

## 8. Security Considerations

Hosts using WebRTC calls will see lot of FW-FLOWDATA attributes. They determine the key by trying a number of candidate keys and

seeing if one of them is correct. The attack works when the keys have low entropy, such as a word from the dictionary. This attack can be mitigated by using strong keys with large entropy. In situations where even stronger mitigation is required, the keys can be dynamically changed using GDOI. The WebRTC server controls how long a firewall session is kept open via the Lifetime value and WebRTC server could use different Lifetime values depending on the anticipated level of trust of the device (e.g. company provided laptop might be trusted more than a Bring Your Own Device (BYOD)); the device with more trust need to obtain its authentication attribute less often). Firewalls in addition to timestamp checking can also maintain a cache of used Nonces, IP source addresses as an effective countermeasure against replay attacks.

All the security considerations applicable to STUN [RFC5389] and ICE [RFC5245] are applicable to this document as well.

## 9. IANA Considerations

Allocate new STUN attribute value for FW-FLOWDATA from the [STUN-ATTR] registry.

## 10. Acknowledgements

The authors would like to thank Prashanth Patil and Ramesh Nethi for review comments.

## 11. References

### 11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

- [RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.
- [RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation", RFC 6407, October 2011.

## 11.2. Informative References

- [I-D.ietf-rtcweb-use-cases-and-requirements]  
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements",  
draft-ietf-rtcweb-use-cases-and-requirements-09 (work in progress), June 2012.
- [STUN-ATTR]  
IANA, "IANA: STUN Attributes", December 2011, <<http://www.iana.org/assignments/stun-parameters/stun-parameters.xml#stun-parameters-3>>.

## Authors' Addresses

Tirumaleswar Reddy  
Cisco Systems, Inc.  
Cessna Business Park, Varthur Hobli  
Sarjapur Marathalli Outer Ring Road  
Bangalore, Karnataka 560103  
India

Email: [tiredy@cisco.com](mailto:tiredy@cisco.com)

Muthu Arul Mozhi Perumal  
Cisco Systems, Inc.  
Cessna Business Park, Varthur Hobli  
Sarjapur Marathalli Outer Ring Road  
Bangalore, Karnataka 560103  
India

Email: [mperumal@cisco.com](mailto:mperumal@cisco.com)

Ram Mohan R  
Cisco Systems, Inc.  
Cessna Business Park, Varthur Hobli  
Sarjapur Marathalli Outer Ring Road  
Bangalore, Karnataka 560103  
India

Email: rmohanr@cisco.com

Dan Wing  
Cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, California 95134  
USA

Email: dwing@cisco.com



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: November 17, 2012

M. Westerlund  
B. Burman  
Ericsson  
May 16, 2012

Codec Control for WebRTC  
draft-westerlund-rtcweb-codec-control-00

Abstract

This document proposes how WebRTC should handle media codec control between peers. With media codec control we mean such parameters as video resolution and frame-rate. This includes both initial establishment of capabilities using the SDP based JSEP signalling and during ongoing real-time interactive sessions in response to user and application events. The solution uses SDP for initial boundary establishment that are rarely, if ever changed. During the session the RTCP based Codec Operations Point (COP) signaling solution is used for dynamic control of parameters enabling timely and responsive controls.

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 <http://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 November 17, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Definitions . . . . .	3
2.1. Abrevations . . . . .	3
2.2. Requirements Language . . . . .	4
3. Overview . . . . .	4
4. Requirements and Motivations . . . . .	5
4.1. Use Cases and Requirements . . . . .	5
4.2. Motivations . . . . .	11
4.2.1. Performance . . . . .	11
4.2.2. Ease of Use . . . . .	13
5. SDP Usage . . . . .	14
6. COP Usage . . . . .	14
7. IANA Considerations . . . . .	15
8. Security Considerations . . . . .	15
9. Acknowledgements . . . . .	16
10. References . . . . .	16
10.1. Normative References . . . . .	16
10.2. Informative References . . . . .	17
Authors' Addresses . . . . .	17

## 1. Introduction

In WebRTC there exist need for codec control to improve the efficiency and user experience of its real-time interactive media transported over a PeerConnection. The fundamentals of the codec control is that the media receiver provides preference for how it would like the media to be encoded to best suit the receiver's consumption of the media stream. This includes parameters such as video resolution and frame-rate, and for audio number of channels and audio bandwidth. It also includes non media specific properties such as how to provision available transmission bit-rates between different media streams.

This document proposes a specific solution for how to accomplish codec control that meets the goals and requirements. It is based on establishing the outer boundaries, when it comes to codec support and capabilities, at PeerConnection establishment using JSEP [I-D.ietf-rtcweb-jsep] and SDP [RFC4566]. During an ongoing session the preferred parameters are signalled using the Codec Operation Point RTCP Extension (COP) [I-D.westerlund-avtext-codec-operation-point]. The java script Application will primarily make its preferences made clear through its usage of the media elements, like selecting the size of the rendering area for video. But it can also use the constraints concept in the API to indicate preferences that the browser can weigh into its decision to request particular preferred parameters.

This document provides a more detailed overview of the solution. Then it discusses the use cases and requirements that motivates the solution, followed by an analysis of the benefits and downsides of the proposed solution. This is followed by a proposed specification of how WebRTC should use SDP and COP.

## 2. Definitions

### 2.1. Abrevations

The following Abbreviations are used in this document.

COP: Codec Operation Point RTCP Extension, the solution for codec control defined in [I-D.westerlund-avtext-codec-operation-point].

JSEP: Java script Session Establishment Protocol  
[I-D.ietf-rtcweb-jsep].

RTP: Real-time Transport Protocol [RFC3550].

SDP: Session Description Protocol [RFC4566].

## 2.2. Requirements Language

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

## 3. Overview

The basic idea in this proposal is to use JSEP to establish the outer limits for behavior and then use Codec Operation Point (COP) [I-D.westerlund-avtext-codec-operation-point] proposal to handle dynamic changes during the session.

Boundary conditions are typically media type specific and in some cases also codec specific. Relevant for video are highest resolution, frame-rate and maximum complexity. These can be expressed in JSEP SDP for H.264 using the H.264 RTP payload format [RFC6184] specifying the profile and level concept. The authors expect something similar for the VP8 payload format [I-D.ietf-payload-vp8].

During the session the browser implementation detects when there is need to use COP to do any of the following things.

- a. Request new target values for codec operation, for example based on that the GUI element displaying a video has changed due to window resize or purpose change. This includes parameters such as resolution, frame-rate, and picture aspect ratio.
- b. Change parameters due to changing display screen attached to the device. Affected parameters include resolution, picture aspect ratio and sample aspect ratio.
- c. Indicate when the end-point changes encoding parameters in its role as sender.
- d. Change important parameters affecting the transport for media streams such as a maximum media bit-rate, token bucket size (to control the burstiness of the sender), used RTP payload type, maximum RTP packet size, application data unit Aggregation (to control amount of audio frames in the same RTP packet).

- e. Affect the relative prioritization of media streams.

The receiving client may send a COP request in RTCP to request some set of parameters to be changed according to the receiving client's preferences. The applications preferences are primarily indicated through its usage of the media elements. But there exist cases and properties where the application will have to provide additional preference information for example using the constraints. The browser implementation takes all these information into account when expressing preference using a set of parameters.

The media sender evaluates the request and weights it against other potential receiver's requests and may update one or more (if scalability is supported) codec operation points to better suit the receivers. Any new operation point(s) are announced using a COP Notification. Independently if the codec operation point(s) are changed or not, the COP request is acknowledged using a COP status message.

Using RTCP and "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124] the COP message can in most cases be sent immediately or with a very small delay. As the message travels in the media plane it will reach the peer or the next middlebox that are part of the media path directly.

#### 4. Requirements and Motivations

This section discusses the use cases and the requirements for codec control. This includes both the ones explicitly discussed in the use case document but also derived ones. This is followed by a discussion why the proposed mechanism is considered the most suitable for WebRTC.

##### 4.1. Use Cases and Requirements

There are use cases and derived requirements in "Web Real-Time Communication Use-cases and Requirements" [I-D.ietf-rtcweb-use-cases-and-requirements].

A Selection of interesting Use Cases and the description parts that are most applicable to Codec Control are:

- 4.2.1 - Simple Video Communication Service: Two or more users have loaded a video communication web application into their browsers, provided by the same service provider, and logged into the service it provides. The web service publishes information about user login status by pushing updates to the web application in the

browsers. When one online user selects a peer online user, a 1-1 video communication session between the browsers of the two peers is initiated. The invited user might accept or reject the session.

During session establishment a self-view is displayed, and once the session has been established the video sent from the remote peer is displayed in addition to the self-view. During the session, each user can select to remove and re-insert the self-view as often as desired. Each user can also change the sizes of his/her two video displays during the session. Each user can also pause sending of media (audio, video, or both) and mute incoming media

The two users may be using communication devices of different makes, with different operating systems and browsers from different vendors.

- 4.2.10 - Multiparty video communication: In this use-case is the Simple Video Communication Service use-case (Section 4.2.1) is extended by allowing multiparty sessions. No central server is involved - the browser of each participant sends and receives streams to and from all other session participants. The web application in the browser of each user is responsible for setting up streams to all receivers.

In order to enhance intelligibility, the web application pans the audio from different participants differently when rendering the audio. This is done automatically, but users can change how the different participants are placed in the (virtual) room. In addition the levels in the audio signals are adjusted before mixing.

Another feature intended to enhance the use experience is that the video window that displays the video of the currently speaking peer is highlighted.

Each video stream received is by default displayed in a thumbnail frame within the browser, but users can change the display size.

Note: What this use-case adds in terms of requirements is capabilities to send streams to and receive streams from several peers concurrently, as well as the capabilities to render the video from all received streams and be able to spatialize, level adjust and mix the audio from all received streams locally in the browser. It also adds the capability to measure the audio level/activity.

- 4.3.3 - Video conferencing system with central server: An organization uses a video communication system that supports the establishment of multiparty video sessions using a central conference server.

The browser of each participant send an audio stream (type in terms of mono, stereo, 5.1, ... depending on the equipment of the participant) to the central server. The central server mixes the audio streams (and can in the mixing process naturally add effects such as spatialization) and sends towards each participant a mixed audio stream which is played to the user.

The browser of each participant sends video towards the server. For each participant one high resolution video is displayed in a large window, while a number of low resolution videos are displayed in smaller windows. The server selects what video streams to be forwarded as main- and thumbnail videos respectively, based on speech activity. As the video streams to display can change quite frequently (as the conversation flows) it is important that the delay from when a video stream is selected for display until the video can be displayed is short.

Note: This use-case adds requirements on support for fast stream switches F7. There exist several solutions that enable the server to forward one high resolution and several low resolution video streams: a) each browser could send a high resolution, but scalable stream, and the server could send just the base layer for the low resolution streams, b) each browser could in a simulcast fashion send one high resolution and one low resolution stream, and the server just selects or c) each browser sends just a high resolution stream, the server transcode into low resolution streams as required.

The derived requirements that applies to codec control are:

- F3: Transmitted streams MUST be rate controlled.
- F6: The browser MUST be able to handle high loss and jitter levels in a graceful way.
- F7: The browser MUST support fast stream switches.
- F24: The browser MUST be able to take advantage of capabilities to prioritize voice and video appropriately.

F25: The browser SHOULD use encoding of streams suitable for the current rendering (e.g. video display size) and SHOULD change parameters if the rendering changes during the session.

It might not be obvious how some of the above requirements actually have impact on the question of controlling the media encoder in a transmitter so let's go through what the document authors consider be its applicability. But let's start with reviewing the topologies that exist.

Peer to Peer: This is the basic topology used in use case "Simple Video Communication Service". Two end-points communicating directly with each other. A PeerConnection directly connects the source and the sink of the media stream. Thus in this case it is simple and straightforward to feed preferences from the sink into the source's media encoder to produce the best possible match that the source is capable of, given the preferences.

Peer to Multiple Peers: A given source have multiple PeerConnections going from the source to a number of receivers, i.e. sinks as described by use case "Multiparty video communication". In some implementations this will be implemented as Peer to Peer topology where only the source for the raw media is common between the different PeerConnections. On more resource constrained devices that can't afford individual media encodings for each PeerConnection the media stream is to be delivered over, there exist a need to merge the different preferences from the different receivers into a single or a set of fewer configurations that can be produced. For codecs that has scalability features, it might be possible to produce multiple actual operation points in a single encoding and media stream. For example multiple frame rates can be produced by H.264 by encoding using a frame structure where some frames can be removed to produce a lower bit-rate and lower frame rate version of the stream. Thus possibly allowing multiple concurrent operation points to be produced to meet the demands for an even larger number of preferred operation points.

Centralized Conferencing: This topology consists of a central server to which each conference participant connects his PeerConnection(s). Over that PeerConnection the participant will receive all the media streams the conference service thinks should be sent and delivered. The actual central node can work in several different modes for media streams. It can be a very simple relay node (RTP transport translator [RFC5117]), where it forwards all media streams arriving to it to the other participants, forming a common RTP session among all participants with full visibility. Another mode of operation would be an RTP mixer that forwards selected media streams using a set of SSRC the

RTP mixer has. The third mode is to perform actual media mixing such as where audio is mixed and video is composited into a new video image and encoded again.

This results in two different behaviors in who needs to merge multiple expressed preferences. For a simple relay central node, the merge of preferences may be placed on the end-point, similar to the resource constrained peer to multiple peer case above. The second alternative is to let the central node merge the preferences into a single set of preferences, which is then signalled to the media source end-point.

Note: In the above it might be possible to establish multiple PeerConnections between an end-point and the central node. The different PeerConnections would then be used to express different preferences for a given media stream. This enables simulcast delivery to the central node so that it can use more than a single operation point to meet the preferences expressed by the multiple receiving participants. That approach can improve the media quality for end-points capable of receiving and using a higher media quality, since they can avoid being constrained by the lowest common denominator of a single operation point.

Peer Relayed: This is not based on an explicit use case in the use case document. It is based on a usage that appears possible to support, and for which there has been interest. The topology is that Peer A sources a media stream and sends it over a PeerConnection to B. B in its turn has a PeerConnection to Peer C. B chooses to relay the incoming media stream from A to C. To maintain quality, it is important that B does not decode and re-encode the media stream (transcoding). Thus a case arises where B will have to merge the preferences from itself and C into the preferences it signals to A.

Comments on the applicability of the requirement on the codec control:

F3: This requirement requires rate-control on the media streams. There will also be multiple media streams being sent to or from a given end-point. Combined, this creates a potential issue when it comes to prioritization between the different media streams and what policy to use to increase and decrease the bit-rate provided for each media stream. The application's preferences combined with other properties such as current resolution and frame-rate affects which parameter that is optimal to use when bit-rate needs to be changed. The other aspect is if one media stream is less relevant so that reducing that stream's quality or even terminating the transmission while keeping others unchanged is the

best choice for the application. In other cases, applying the cheese cutter principle and reduce all streams in equal proportion is the most desirable. Another aspect is the potential for requesting aggregation of multiple audio frames in the same RTP packet to reduce the overhead and thus lower the bit-rate for some increased delay and packet loss sensitivity.

F6: The browser MUST be able to handle high loss and jitter levels in a graceful way. When such conditions are encountered, it will be highly beneficial for the receiver to be able to indicate that the sender should try to combat this by changing the encoding and media packetization. For example for audio it might be beneficial to aggregate several frames together and apply additional levels of FEC on those fewer packets that are produced to reduce the residual audio frame loss.

F7: The browser MUST support fast stream switches. Fast stream switches occur in several ways in WebRTC. One is in the centralized conferencing when relay based central nodes turn on and off individual media streams depending on the application's current needs. Another is RTP mixers that switches input sources for a given outgoing SSRC owned by the mixer. This should have minimal impact on a receiver as there is no SSRC change. Along the same lines, the application can cause media stream changes by changing their usage in the application. By changing the usage of a media stream from being the main video to become a thumbnail of one participant in the session, there exist a need to rapidly switch the video resolution to enable high efficiency and avoid increased bit-rate usage.

F24: The browser MUST be able to take advantage of capabilities to prioritize voice and video appropriately. This requirement comes from the QoS discussion present in use case 4.2.6 (Simple Video Communication Service, QoS). This requirement assumes that the application has a preference for one media type over another. Given this assumption, the same prioritization can actually occur for a number of codec parameters when there exist multiple media streams and one can individually control these media streams. This is another aspect of the discussion for requirement F3.

F25: The browser SHOULD use encoding of streams suitable for the current rendering (e.g. video display size) and SHOULD change parameters if the rendering changes during the session. This requirement comes from a very central case that a receiving application changes the display layout and where it places a given media stream. Thus changing the amount of screen estate that the media stream is viewed on also changes what resolution that would be the optimal to use from the media sender. However, this

discussion should not only apply to video resolution. Additional application preferences should result in preferences being expressed to the media sender also for other properties, such as video frame-rate. For audio, number of audio channels and the audio bandwidth are relevant properties.

The authors hope this section has provided a sufficiently clear picture that there exist both multiple topologies with different behaviors, and different points where preferences might need to be merged. The discussion of the requirements also provides a view that there are multiple parameters that needs to be expressed, not only video resolution.

#### 4.2. Motivations

This section discusses different types of motivations for this solution. It includes comparison to the solution described in "RTCWEB Resolution Negotiation" [I-D.alvestrand-rtcweb-resolution].

##### 4.2.1. Performance

The proposed solution has the following performance characteristics. The initial phase, establishing the boundaries, is done in parallel with the media codec negotiation and establishment of the PeerConnection. Thus using the signalling plane is optimal as this process does not create additional delay or significant overhead.

During an ongoing communication session, using COP messages in RTCP has the following properties:

**Path Delay:** The COP messages are contained in the RTCP packets being sent over the PeerConnection, i.e. the most optimal peer to peer path that ICE has managed to get to work. Thus one can expect this path to be equal or shorter in delay than the signalling path being used between the PeerConnection end-points. If the signalling message is instead sent over the PeerConnection's data channel, it will be using the same path. In almost all cases, the direct path between two peers will also be shorter than a path going via the webserver.

**Media Plane:** The COP messages will always go to the next potential RTP/RTCP processing point, i.e. the one on the other side of the PeerConnection. Even for multiparty sessions using centralized servers, the COP message may need to be processed in the middle to perform the merger of the different participant's preferences.

Overhead: An RTCP COP message can be sent as reduced size RTCP message [RFC5506] thus having minimal unnecessary baggage. For example a COP Request message requesting a new target resolution from a single SSRC will be 29 bytes. Using reduced size RTCP keeps the average RTCP size down and enables rapid recovery of the early allowed flag in early mode and in more cases enable the immediate mode.

Minimal Blocking: Using RTCP lets the transmission of COP messages be governed by RTCP's transmission rules. As WebRTC will be using the SAVPF profile it is possible to use the early mode, allowing an early transmission of an RTCP packet carrying a feedback event, like a COP request, to be sent with little delay. It might even be possible to determine that the immediate mode of operation can be enabled, thus allowing the RTCP feedback events to be sent immediate in all cases while using the mode. The small overhead and usage of reduced size RTCP will help ensure that the times when transmission of a COP message will be blocked is a rare event and will quickly be released.

The next aspect of RTCP blocking is that we expect that the application will need to rapidly switch back and forth between codec parameters. Thus requiring both a protocol that allows quick setting of parameters and also the possibility to revert back to previous preferences while the request is outstanding. COP has support for such updated requests, even if the first request is in flight.

If the above is compared to the properties that Harald Alvestrand's proposal [I-D.alvestrand-rtcweb-resolution] has, the following differences are present. When it comes to signalling path delay, a signalling plane based solution will in almost all cases at best have the same path delay as a media plane solution, achieved by using the data channel to carry the signalling. There the only difference will be the message size, which will only incur a minor difference in transfer times. But in cases where the application has not implemented use of the data channel, the signalling path will be worse, possibly significantly.

Using the signalling plane for solutions based on centralized conference mixers can easily result in that the request message needs to be processed in the webserver before being forwarded to the mixer node that actually processes the media, followed by the central mixer triggering additional signalling messages to other end-points that also needs to react. This can be avoided assuming that the data channel is used for signalling transport. Using the media plane for such signalling will be equal or better in almost all cases.

When it comes to blocking, there exist a significant issue with using JSEP to carry this type of messages. Someone that has sent an SDP offer in an offer/answer exchange is blocked from sending a new update until it has received a final or provisional answer to that offer. Here COP has a great advantage as the design has taken rapid change of parameters into consideration and allows multiple outstanding requests.

#### 4.2.2. Ease of Use

We see a great benefit in that COP can be allowed to be mainly driven by the browser implementation and its knowledge of how media elements are currently being used by the application. For example the video resolution of the display area can be determined by the browser, further determining that the resource consumption would be reduced and the image quality improved or at least maintained by requesting another target resolution better suiting the current size. There are also other metrics or controls that exist in the browser space, like the congestion control that can directly use the COP signalling to request more suitable parameters given the situation.

Certain application preferences can't be determined based solely on the usage. Thus using the constraints mechanism to indicate preferences is a very suitable solution for most such properties. For example the relative priority of media streams, or a desire for lower frame rate to avoid reductions in resolution or image quality SNR are likely to need constraints.

This type of operation results in better performance for simple applications where the implementor isn't as knowledgeable about the need to initiate signalling to trigger a change of video resolution. Thus providing good performance in more cases and having less amount of code in their applications.

Still more advanced applications should have influence on the behavior. This can be realized in several ways. One is to use the constraints to inform the browser about the application's preferences how to treat the different media streams, thus affecting how COP is used. If required, it is possible to provide additional API functionalities for the desired controls.

The authors are convinced that providing ease of use for the simple application is important. Providing more advanced controls for the advanced applications is desirable.

## 5. SDP Usage

SDP SHALL be used to establish boundaries and capabilities for the media codec control in WebRTC. This includes the following set of capabilities that is possible to express in SDP:

Codec Capabilities: For all media codecs it is needed to determine what capabilities that are available if there exist optional functionalities. This concerns both media encoding and the RTP payload format as codec control can affect both. For codecs where the span of complexities are large there might exist need to express the level of complexity supported. For Video codecs like H.264 this can be expressed by the profile level ID. These capabilities are expected to be defined by the RTP payload format or in SDP attributes defined in the RTP payload formats to be used.

COP Parameters Supported: SDP SHALL be used to negotiate the set of COP parameters that the peers can express preferences for and for which they will send notification on their sets of parameter values used.

## 6. COP Usage

An WebRTC end-point SHALL implement Codec Operation Point RTCP Extension [I-D.westerlund-avtext-codec-operation-point]. The following COP parameters SHALL be supported:

- o Payload Type
- o Bitrate
- o Token Bucket Size
- o Framerate
- o Horizontal Pixels
- o Vertical Pixels
- o Maximum RTP Packet Size
- o Maximum RTP Packet Rate
- o Application Data Unit Aggregation

Please note that also the ALT and ID parameters must be implemented

in COP for COP to correctly function.

To make COP usage efficient the end-point SHALL implement Reduced size RTCP packets [RFC5506].

To provide in addition to requesting specific frame-rates also the RTCP Codec Control Messages "Temporal-Spatial Trade-off Request and Notification" [RFC5104] . This enables a receiver to make a relative indication of their preferred trade-off between spatial and temporal quality. This provides an highly useful indication to the media sender about what the receiver prefer in a relative sense. The COP framerate or resolution parameters can be used to further provides target, max or min values to further indicate within which set of parameters the sender should find this relative trade-off.

To enable an receiver to temporarily halt or pause delivery of a given media stream an WebRTC end-point SHALL also implement "RTP Media Stream Pause and Resume" [I-D.westerlund-avtext-rtp-stream-pause]. This is important COP related features as described by the use case and motivations to enable the receiver to indicate that it prefers to have a given media stream halted if the aggregate media bit-rate is reduced. It can also be used to recover aggregate media bit-rate when the application has no current use of a given media stream, but may rapidly need it again due to interactions in the application or with other instances.

## 7. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 8. Security Considerations

The usage of COP and its security issues are described in [I-D.westerlund-avtext-codec-operation-point]. The main threats to this usage of COP are the following things:

- a. That the SDP based codec boundary signalling and COP parameter negotiation could be intercepted and modified. Thus enabling denial of service attacks on the end-points reducing the scope of the COP usage and the media codec parameters to provide sub-optimal quality or block certain features. To prevent this the SDP needs to be authenticated and integrity protected.

- b. The COP messages themselves could be modified to affect the negotiated codec parameters. This could have severe impact on the media quality as media streams can be completely throttled, or configured to very reduced framerate or resolution. To prevent this source authentication and integrity protection must be applied to the RTCP compound packets.
- c. In multi-party applications of COP an entity may need to combine multiple sets of requested parameters. In these multi-party cases a particular participant may target the other participants and actively try to degrade their experience. Any COP entity merging sets will need to consider if a particular participant is actively harmful to the others and can choose to ignore that entity's request.

## 9. Acknowledgements

## 10. References

### 10.1. Normative References

- [I-D.ietf-rtcweb-jsep]  
Uberti, J. and C. Jennings, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-00 (work in progress), March 2012.
- [I-D.westerlund-avtext-codec-operation-point]  
Westerlund, M., Burman, B., and L. Hamm, "Codec Operation Point RTCP Extension", draft-westerlund-avtext-codec-operation-point-00 (work in progress), March 2012.
- [I-D.westerlund-avtext-rtp-stream-pause]  
Akram, A., Burman, B., Grondal, D., and M. Westerlund, "RTP Media Stream Pause and Resume", draft-westerlund-avtext-rtp-stream-pause-01 (work in progress), May 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session

Description Protocol", RFC 4566, July 2006.

- [RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, February 2008.
- [RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

## 10.2. Informative References

- [I-D.alvestrand-rtcweb-resolution]  
Alvestrand, H., "RTCWEB Resolution Negotiation",  
draft-alvestrand-rtcweb-resolution-00 (work in progress),  
April 2012.
- [I-D.ietf-payload-vp8]  
Westin, P., Lundin, H., Glover, M., Uberti, J., and F.  
Galligan, "RTP Payload Format for VP8 Video",  
draft-ietf-payload-vp8-04 (work in progress), March 2012.
- [I-D.ietf-rtcweb-use-cases-and-requirements]  
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-  
Time Communication Use-cases and Requirements",  
draft-ietf-rtcweb-use-cases-and-requirements-07 (work in  
progress), April 2012.
- [RFC5117] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 5117,  
January 2008.
- [RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for  
Real-time Transport Control Protocol (RTCP)-Based Feedback  
(RTP/SAVPF)", RFC 5124, February 2008.
- [RFC6184] Wang, Y., Even, R., Kristensen, T., and R. Jesup, "RTP  
Payload Format for H.264 Video", RFC 6184, May 2011.

Authors' Addresses

Magnus Westerlund  
Ericsson  
Farogatan 6  
SE-164 80 Kista  
Sweden

Phone: +46 10 714 82 87  
Email: magnus.westerlund@ericsson.com

Bo Burman  
Ericsson  
Farogatan 6  
SE-164 80 Kista  
Sweden

Phone: +46 10 714 13 11  
Email: bo.burman@ericsson.com

