

Network Working Group
Internet-Draft
Updates: 3515 (if approved)
Intended status: Standards Track
Expires: April 30, 2015

R. Sparks
Oracle
A. Roach
Mozilla
October 27, 2014

Clarifications for the use of REFER with RFC6665
draft-sparks-sipcore-refer-clarifications-05

Abstract

An accepted SIP REFER method creates an implicit subscription using the SIP-Specific Event Notification Framework. That framework was revised by RFC6665. This document highlights the implications of the requirement changes in RFC6665, and updates the definition of the REFER method, RFC3515, to clarify and disambiguate the impact of those changes.

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 30, 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. Conventions and Definitions	2
2. Introduction	2
3. Use of GRUU is mandatory	2
4. Dialog reuse is prohibited	3
5. Security Considerations	3
6. IANA Considerations	4
7. References	4
7.1. Normative References	4
7.2. Informative References	4
Authors' Addresses	5

1. Conventions and Definitions

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. Introduction

An accepted SIP REFER method creates an implicit subscription using the SIP-Specific Event Notification Framework. That framework was revised by [RFC6665]. This document highlights the implications of the requirement changes in RFC6665, and updates [RFC3515] to clarify and disambiguate the impact of those changes.

3. Use of GRUU is mandatory

Section 4.5.1 of [RFC6665] makes GRUU [RFC5627] mandatory to implement and use as the local target in the subscription created by the REFER request.

A user agent constructing any REFER that can result in an implicit subscription MUST populate its Contact header field with a GRUU.

As RFC6665 details, this is necessary to ensure that NOTIFY requests sent in the implicitly created subscription arrive at this user agent without creating a second usage inside an existing dialog. Using the "norefersub" option tag [RFC4488] does not change this requirement, even if used in a "Require" header field. Even if the recipient supports the "norefersub" mechanism, and accepts the request with the option tag in the "Require" header field, it is allowed to return a "Refer-Sub" header field with a value of "true" in the response, and create an implicit subscription.

A UA that will accept a REFER request needs to include a GRUU in the Contact header field of all INVITE requests. This ensures that out-of-dialog REFER requests corresponding to any resulting INVITE dialogs arrive at this UA. Future extensions (such as [I-D.sparks-sipcore-refer-explicit-subscription]) might relax this requirement by defining a REFER request that cannot create an implicit subscription.

4. Dialog reuse is prohibited

If a peer in an existing dialog has provided a GRUU as its Contact, sending a REFER that might result in an additional dialog usage within that dialog is prohibited. This is a direct consequence of [RFC6665] requiring the use of GRUU, and the requirements in section 4.5.2 of that document.

A user agent constructing a REFER request that could result in an implicit subscription MUST build it as an out-of-dialog message as defined in [RFC3261]. Thus, the REFER request will have no tag parameter in its To: header field.

A user agent wishing to identify an existing dialog (such as for call transfer as defined in [RFC5589] MUST use the "Target-Dialog" extension defined in [RFC4538] to do so, and user agents accepting REFER MUST be able to process that extension in requests they receive.

If a user agent can be certain that no implicit subscription will be created as a result of sending a REFER request (such as by requiring an extension that disallows any such subscription), the REFER request MAY be sent within an existing dialog. Such a REFER will be constructed with its Contact header field populated with the dialog's Local URI as specified in section 12 of [RFC3261].

As described in section 4.5.2 of [RFC6665], there are cases where a user agent may fall back to sharing existing dialogs for backwards-compatibility purposes. This applies to REFER only when the peer has not provided a GRUU as its Contact in the existing dialog (i.e. when the peer is a pre-RFC6665 implementation).

5. Security Considerations

This document introduces no new security considerations directly. The updated considerations in [RFC6665] apply to the implicit subscription created by an accepted REFER request.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.
- [RFC4538] Rosenberg, J., "Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)", RFC 4538, June 2006.
- [RFC5627] Rosenberg, J., "Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)", RFC 5627, October 2009.
- [RFC6665] Roach, A., "SIP-Specific Event Notification", RFC 6665, July 2012.

7.2. Informative References

- [I-D.sparks-sipcore-refer-explicit-subscription] Sparks, R., "Explicit Subscriptions for the REFER Method", draft-sparks-sipcore-refer-explicit-subscription-00 (work in progress), June 2014.
- [RFC4488] Levin, O., "Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription", RFC 4488, May 2006.
- [RFC5589] Sparks, R., Johnston, A., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, June 2009.

Authors' Addresses

Robert Sparks
Oracle
7460 Warren Parkway
Suite 300
Frisco, Texas 75034
US

Email: rjsparks@nostrum.com

Adam Roach
Mozilla
Dallas, TX
US

Phone: +1 650 903 0800 x863
Email: adam@nostrum.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2015

R. Sparks
Oracle
October 21, 2014

Explicit Subscriptions for the REFER Method
draft-sparks-sipcore-refer-explicit-subscription-02

Abstract

The Session Initiation Protocol (SIP) REFER request, as defined by RFC3515, triggers an implicit SIP-Specific Event Notification framework subscription. Conflating the start of the subscription with handling the REFER request makes negotiating SUBSCRIBE extensions impossible, and complicates avoiding SIP dialog sharing. This document defines extensions to REFER to remove the implicit subscription and, if desired, replace it with an explicit one.

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 24, 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. Conventions and Definitions	2
2. Introduction	3
3. Overview	3
3.1. Explicit Subscriptions	3
3.2. No Subscriptions	4
4. The Explicit Subscription Extension	5
4.1. Sending a REFER	5
4.2. Processing a REFER Response	5
4.3. Processing a Received REFER	5
4.4. Subscribing to the 'refer' Event	6
4.5. Processing a Received SUBSCRIBE	7
4.6. Sending a NOTIFY	7
4.7. Managing 'refer' Event State	7
4.8. The Refer-Events-At Header Field	8
5. The No Subscription Extension	8
5.1. Sending a REFER	8
5.2. Processing a REFER Response	8
5.3. Processing a Received REFER	9
6. The 'explicitsub' and 'nosub' Option Tags	9
7. Updates to RFC 3515	10
8. Security Considerations	10
9. IANA Considerations	11
9.1. Register the 'explicitsub' Option Tag	11
9.2. Register the 'nosub' Option Tag	11
9.3. Register the 'Refer-Events-At' Header Field	11
10. Changelog	12
10.1. 01 to 02	12
10.2. 00 to 01	12
11. References	12
11.1. Normative References	12
11.2. Informative References	13
Author's Address	13

1. Conventions and Definitions

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. Introduction

REFER as defined by [RFC3515] triggers an implicit SIP-Specific Event Framework subscription. Sending a REFER within a dialog established by an INVITE results in dialog reuse and the associated problems described in [RFC5057]. The SIP-Specific Event Notification framework definition [RFC6665] disallows such dialog reuse. Call transfer, as defined in [RFC5589], thus requires sending a REFER request on a new dialog, associating it with an existing dialog using the 'Target-Dialog' mechanism defined in [RFC4538].

Because there is no explicit SUBSCRIBE request, the tools for negotiating subscription details are unavailable for REFER subscriptions. This includes negotiating subscription duration and providing information through Event header field parameters. The use of the SIP 'Supported' and 'Require' extension mechanisms [RFC3261] is complicated by the implicit subscription. It is unclear whether the extension applies to handling the REFER request itself, or to the messages in the subscription created by the REFER, or both. Avoiding this confusion requires careful specification in each extension. Existing extensions do not provide this clarity.

This document defines two mechanisms that remove the implicit subscription, one of which replaces it with an explicit one. The benefits of doing so include:

- o Allowing REFER to be used within INVITE-created dialogs without creating dialog reuse.
- o Allowing standard subscription parameter negotiation.
- o Allowing standard negotiation of SIP extensions.

There are limitations on when it is appropriate to use the extension that allows an explicit subscription, related directly to definition of non-INVITE transaction handling SIP. These limitations are discussed in Section 4.1.

3. Overview

This section provides a non-normative overview of the behaviors defined in subsequent sections.

3.1. Explicit Subscriptions

A SIP User-Agent (UA) that wishes to issue a REFER request that will not create an implicit subscription, but will allow an explicit one, will include a new option tag, 'explicitsub', in the Require header

field of the REFER request. This REFER could be sent either within an existing dialog, or as an out-of-dialog request.

If the recipient of the REFER accepts the request, it will begin managing the 'refer' event state described in RFC 3515, and will provide a URI that will reach an event server that will service subscriptions to that state. (In many cases, the recipient of the REFER will perform the role of event server itself.) That URI is returned in a new header field in the REFER response named 'Refer-Events-At'.

The UA that issued the REFER can now subscribe to the 'refer' event at the provided URI, using a SUBSCRIBE request with a new dialog identifier. The full range of negotiation mechanisms is available for its use in that request. As detailed in RFC 6665 and RFC 3515, the event server accepting the subscription will send an immediate NOTIFY with the current refer event state, additional NOTIFY messages as the refer state changes, and a terminal NOTIFY message when the referred action is complete. It is, of course, possible that the initial NOTIFY is also the terminal NOTIFY.

It is possible that the referred action is completed before the SUBSCRIBE arrives at the event server. The server needs to retain the final refer event state for some period of time to include in the terminal NOTIFY that will be sent for such subscriptions. It is also possible that a SUBSCRIBE will never arrive.

This extension makes it possible to separate the event server that will handle subscriptions from the UA that accepted the REFER. Such a UA could use mechanisms such as PUBLISH [RFC3903] to convey the refer event state to the event server. This extension also makes it possible to allow more than one subscription to the refer event state.

3.2. No Subscriptions

A UA that wishes to issue a REFER request that will not create an implicit subscription, and tell the recipient that it is not interested in creating an explicit subscription, will include a new option tag, 'nosub', in the Require header field of the REFER request. This REFER could be sent either within an existing dialog or as an out-of-dialog request.

If the recipient of the REFER accepts the request, it knows not to create an implicit subscription, and that no explicit subscription will be forthcoming. The recipient will continue to process the request indicated in the Refer-To header field as specified in RFC

3515, but it can avoid the cost of preparing to handle any subscriptions to the state of handling that request.

4. The Explicit Subscription Extension

4.1. Sending a REFER

To suppress the creation of any implicit subscription, and allow for an explicit one, a UA forming a REFER request will include the option tag 'explicitsub' in the "Require" header field of the request. The REFER request is otherwise formed following the requirements of [RFC3515]. Since this REFER has no chance of creating an implicit subscription, the UA MAY send the REFER request within an existing dialog or out-of-dialog.

Note that if the REFER forks (see [RFC3261]), only one final response will be returned to the issuing UA. If it is important that the UA be able to subscribe to any refer state generated by accepting this request, the request needs to be formed to limit the number of places that it will be accepted to one. This can be achieved by sending the REFER request within an existing dialog, or by using the Target-Dialog mechanism defined in [RFC4538]. If it is possible for the request to be accepted in more than one location, and things would go wrong if the UA did not learn about each location that the request was accepted, using this extension is not appropriate.

4.2. Processing a REFER Response

The UA will process responses to the REFER request as specified in [RFC3515] (and, consequently, [RFC3261]). In particular, if the REFER was sent to an element that does not support or is unwilling to use this extension, the response will contain a 420 Bad Extension response code (see section 8.1.3.5 of [RFC3261]). As that document states, the UA can retry the request without using this extension.

If the UA receives a 2xx-class response, it will contain a Refer-Events-At header field (Section 4.8) with a single URI as its value. If the UA is interested in the state of the referenced action, it will subscribe to the 'refer' event at that URI.

4.3. Processing a Received REFER

An element receiving a REFER request requiring the 'explicitsub' extension will use the same admissions policies that would be used without the extension, with the addition that it is acceptable to admit an in-dialog REFER request requiring this extension since it can not create another usage inside that dialog. In particular, see section 5.2 of [RFC3515].

Accepting a REFER request that requires 'explicitsub' does not create a dialog, or a new usage within an existing dialog. The element MUST NOT create an implicit subscription when accepting the REFER request.

An element that accepts a REFER request with 'explicitsub' in its Require header field MUST return a 200 response containing a sip: or sips: URI that can be used to subscribe to the refer event state associated with this REFER request. This URI MUST uniquely identify this refer event state. The URI needs to reach the event server when used in a SUBSCRIBE request from the element that sent the REFER. One good way to ensure the URI provided has that property is to use a GRUU [RFC5627] for the event server. As discussed in Section 8, possession of this URI is often the only requirement for authorizing a subscription to it. Implementations may wish to provide a URI constructed in a way that is hard to guess. Again, using a GRUU is one good way to achieve this property.

The accepting element will otherwise proceed with the processing defined in [RFC3515].

The event server identified by the Refer-Events-At URI could receive SUBSCRIBE requests at any point after the response containing the Refer-Events-At header is sent. Implementations should take care to ensure the event server is ready to receive those SUBSCRIBE requests before sending the REFER response, but as with all non-INVITE responses, the response should be sent as soon as possible (see [RFC4321]). It is also possible that the referred action may complete before any SUBSCRIBE request arrives. The event server will need to maintain the final refer event state for a period of time after the action completes in order to serve such subscriptions (see Section 4.6).

4.4. Subscribing to the 'refer' Event

A UA that possesses a URI obtained from a Refer-Events-At header field, MAY subscribe to the refer event state at that URI. It does so following the requirements of [RFC6665], placing the token 'refer' in the Event: header field and the URI in the Request-URI of the SUBSCRIBE request. The SUBSCRIBE request MUST NOT reuse any existing dialog identifiers.

Subsequent handling of the subscription MUST follow the requirements of [RFC6665] and [RFC3515]. In particular, as discussed in section 2.4.6, the NOTIFY messages in the subscription might include an id parameter in their Event header fields. Subsequent SUBSCRIBE requests used to refresh or terminate this subscription MUST contain this id parameter. Note that the rationale for the id parameter provided in that section is not relevant when this extension is used.

The URI returned in the Refer-Events-At header field uniquely identifies appropriate state, making the id parameter redundant. However, this behavioral requirement is preserved to reduce the number of changes to existing implementations in order to support this extension, and to make it more likely that existing diagnostic tools will work with little or no modification.

4.5. Processing a Received SUBSCRIBE

An event server receiving a SUBSCRIBE request will process it according to the requirements of [RFC6665]. The event server MAY choose to authorize the SUBSCRIBE request based on the Request-URI corresponding to existing refer event state. It MAY also require further authorization as discussed in Section 8.

When accepting a subscription, the event server will establish the initial subscription duration using the guidance in section 3.4 of [RFC3515].

4.6. Sending a NOTIFY

NOTIFY messages within a subscription are formed and sent following the requirements in [RFC3515]. See, in particular, section 2.4.5 of that document.

4.7. Managing 'refer' Event State

As described in [RFC3515], an element creates the state for event 'refer' when it accepts a REFER request. It updates that state as the referred request proceeds, ultimately reaching a state where the request has completed, and the final state is known.

In RFC 3515 implementations, it was a reasonable design choice to destroy the refer event state immediately after sending the NOTIFY that terminated the implicit subscription. This is not the case when using this extension. It is possible for the referenced request to complete very quickly, perhaps sooner than the time it takes the response to the REFER to traverse the network to the UA that sent the request, and the time it takes that agent to send the SUBSCRIBE request for the event state to the URI the response provides. Thus the event server MUST retain the final refer event state for a reasonable period of time, which SHOULD be at least $2*64*T1$ (that is, 64 seconds), representing an upper-bound estimate of the time it would take to complete two non-INVITE transactions: the REFER, and an immediate SUBSCRIBE.

If an otherwise acceptable SUBSCRIBE arrives during this retention period, the subscription would be accepted, and immediately

terminated with a NOTIFY containing the final event state with a Subscription-State of terminated with a reason value of "noresource".

4.8. The Refer-Events-At Header Field

The 'Refer-Events-At' header field is an extension-header as defined by [RFC3261]. Its ABNF is as follows:

```
Refer-Events-At: "Refer-Events-At" HCOLON
                  LAQUOT ( SIP-URI / SIPS-URI ) RAQUOT
                  * ( SEMI generic-param )
```

See [RFC3261] for the definition of the elements used in that production.

Note that this rule does not allow a full addr-spec as defined in RFC 3261, and it mandates the use of the angle brackets. That is:

```
Refer-Events-At: <sips:vPT3izGmo8NTxaPADRZvEAY22BKx@example.com;gr>
```

is well formed, but

```
Refer-Events-At: sip:wsXa9mkHtPcGu8@example.com
```

is invalid.

The 'Refer-Events-At' header field is only meaningful in a 200 response to a REFER request. If it appears in the header of any other SIP message, its meaning is undefined and it MUST be ignored.

5. The No Subscription Extension

5.1. Sending a REFER

To suppress the creation of any implicit subscription, and signal that no explicit subscription will be forthcoming, a UA forming a REFER request will include the option tag 'nosub' in the "Require" header field of the request. The REFER request is otherwise formed following the requirements of [RFC3515]. Since this REFER has no chance of creating an implicit subscription, the UA MAY send the REFER request within an existing dialog or out-of-dialog.

5.2. Processing a REFER Response

The UA will process responses to the REFER request as specified in [RFC3515] (and, consequently, [RFC3261]). In particular, if the REFER was sent to an element that does not support or is unwilling to use this extension, the response will contain a 420 Bad Extension

response code (see section 8.1.3.5 of [RFC3261]). As that document states, the UA can retry the request without using this extension.

5.3. Processing a Received REFER

An element receiving a REFER request requiring the 'nosub' extension will use the same admissions policies that would be used without the extension, with the addition that it is acceptable to admit an in-dialog REFER request requiring this extension since it can not create another usage inside that dialog. In particular, see section 5.2 of [RFC3515].

Accepting a REFER request that requires 'nosub' does not create a dialog, or a new usage within an existing dialog. The element MUST NOT create an implicit subscription when accepting the REFER request. Furthermore, the element accepting the REFER request is not required to maintain any state for serving refer event subscriptions.

The accepting element will otherwise proceed with the processing defined in [RFC3515].

6. The 'explicitsub' and 'nosub' Option Tags

This document defines the 'explicitsub' option tag, used to signal the use of the extension defined in Section 4, and the 'nosub' option tag, used to signal the use of the extension defined in Section 5.

The use of either option tag in a Require header field is only defined when it appears in a REFER request. A UA MUST NOT include the 'explicitsub' or 'nosub' option tag in the Require header field of any request other than REFER. A UA MUST NOT include the 'explicitsub' or 'nosub' option tag in the Require header field of any SIP response other than a 421 response to a REFER request.

The 'explicitsub' and 'nosub' option tags MAY appear in the Supported header field of SIP messages, and in sip.extensions feature tag defined in [RFC3840]. This signals only that the UA including the value is aware of the extensions. In particular, a UA can only invoke the use of one of the extensions in a request. A User-Agent Server (UAS) that is processing a REFER request that lists 'explicitsub' or 'nosub' in its Supported header field and wishes to use one of those extensions will return a 421 response indicating which extension is required.

OPEN ISSUE: This description of the use of 421 is not yet perfectly aligned with RFC3261's definition.

7. Updates to RFC 3515

The requirement in section 2.4.4 of [RFC3515] to reject out-of-dialog SUBSCRIBE requests to event 'refer' is removed. An element MAY accept a SUBSCRIBE request to event 'refer', following the requirements and guidance in this document. REFER is no longer the only mechanism that can create a subscription to event 'refer'.

[RFC6665] section 8.3.1 deprecates the 202 Accepted response code. New implementations of REFER, whether using the 'explicitsub' extension or not, will never emit a 202 response code. Where RFC 3515 specifies using 202, new implementations MUST use 200 instead.

8. Security Considerations

The considerations of [RFC3515] all still apply to a REFER request using this extension. The considerations there for the implicit subscription apply to any explicit subscription for the 'refer' event.

This update to RFC 3515 introduces a new authorization consideration. An element receiving an initial SUBSCRIBE request to the 'refer' event needs to decide whether the subscriber should be allowed to see the refer event state. In RFC 3515, this decision was conflated with accepting the REFER request, and the only possible subscriber was the element that sent the REFER. With this update, there may multiple subscribers to any given refer event state.

This document allows an element to accept an initial SUBSCRIBE request based on having a Request-URI that identifies existing refer event state. (Such a URI will have previously been sent in the Refer-Events-At header field in a successful REFER response). The element retrieving that URI from the response, and any elements that element shares the URI with are authorized to SUBSCRIBE to the event state. Consequently, the URI should be constructed so that it is not easy to guess, and should be protected against eavesdroppers when transmitted. For instance, SIP messages containing this URI SHOULD be sent using TLS or DTLS. An event server receiving a REFER request over an unprotected transport can redirect the requester to use a protected transport before accepting the request. A good way to ensure that subscriptions use a protected transport is to only construct sips: URIs. The event server can also require any of the additional authorization mechanisms allowed for any SIP request. For example, the event server could require a valid assertion of the subscriber's identity using [RFC4474].

The URI provided in a 'Refer-Events-At' header field will be used as the Request-URI of SUBSCRIBE requests. A malicious agent could take

advantage of being able to choose this URI in ways similar to the ways an agent sending a REFER request can take advantage of the Refer-To URI, as described in the security considerations section of RFC 3515. In particular, the malicious agent could cause a SIP SUBSCRIBE to be sent as raw traffic towards a victim. If the victim is not SIP aware, and the SUBSCRIBE is sent over UDP, there is (at most) a factor of 11 amplification due to retransmissions of the request. The potential for abuse in this situation is lower than that of the Refer-To URI, since the URI can only have a sip: or sips: scheme, and is only provided in a REFER response. A malicious agent would have to first receive a REFER request to take advantage of providing a Refer-Events-At URI.

9. IANA Considerations

9.1. Register the 'explicitsub' Option Tag

The option tag 'explicitsub' is registered in the 'Option Tag' subregistry of the 'Session Initiation Protocol (SIP) Parameters' registry by adding a row with these values:

Name: explicitsub

Description: This option tag identifies an extension to REFER to suppress the implicit subscription, and provide a URI for an explicit subscription.

Reference: (this document)

9.2. Register the 'nosub' Option Tag

The option tag 'nosub' is registered in the 'Option Tag' subregistry of the 'Session Initiation Protocol (SIP) Parameters' registry by adding a row with these values:

Name: nosub

Description: This option tag identifies an extension to REFER to suppress the implicit subscription, and indicate that no explicit subscription is forthcoming.

Reference: (this document)

9.3. Register the 'Refer-Events-At' Header Field

The header field described in Section 4.8 is registered in the 'Header Fields' subregistry of the 'Session Initiation Protocol (SIP) Parameters' registry by adding a row with these values:

Header Name: Refer-Events-At

compact: (none: the entry in this column should be blank)

Reference: (this document)

10. Changelog

RFC Editor - please remove this section when formatting this document as an RFC.

10.1. 01 to 02

1. Added the 'nosub' option tag
2. Added text calling out the limitations on explicitsub when the REFER might be accepted in more than one place.

10.2. 00 to 01

1. Replaced strawman proposal with a formal definition of the mechanism. Added an overview, and detailed security considerations.

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.
- [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.
- [RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.
- [RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.
- [RFC6665] Roach, A., "SIP-Specific Event Notification", RFC 6665, July 2012.

11.2. Informative References

- [RFC3903] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.
- [RFC4321] Sparks, R., "Problems Identified Associated with the Session Initiation Protocol's (SIP) Non-INVITE Transaction", RFC 4321, January 2006.
- [RFC4474] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.
- [RFC4538] Rosenberg, J., "Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)", RFC 4538, June 2006.
- [RFC5057] Sparks, R., "Multiple Dialog Usages in the Session Initiation Protocol", RFC 5057, November 2007.
- [RFC5589] Sparks, R., Johnston, A., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, June 2009.
- [RFC5627] Rosenberg, J., "Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)", RFC 5627, October 2009.

Author's Address

Robert Sparks
Oracle
7460 Warren Parkway
Suite 300
Frisco, Texas 75034
US

Email: rjsparks@nostrum.com

SIPCORE Working Group
Internet-Draft
Intended Status: Standards Track
Expires: April 13, 2015

R. Shekh-Yusef
Avaya
October 10, 2014

Key-Derivation Authentication Scheme
draft-yusef-sipcore-key-derivation-00

Abstract

This document defines a Key-Derivation Authentication Scheme, based on the PBKDF2 Key Derivation Function (KDF), that could be used with the challenge-response authentication framework used by SIP to authenticate the user.

The scheme allows two parties to establish a mutually authenticated communication channel based on a shared password, without ever sending the password on the wire.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License 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	Terminology	3
2	Operations Overview	3
3	The Challenge	5
4	The Response	6
5	The Confirmation	7
6	Security Considerations	7
7	IANA Considerations	7
8.	Acknowledgments	7
9	References	8
9.1	Normative References	8
9.2	Informative References	8
	Authors' Addresses	8

1 Introduction

SIP [RFC3261] uses the Digest Authentication schemes with the general framework for access control and authentication, which is used by a server to challenge a client request and by a client to provide authentication information.

The challenge-response framework relies on passwords chosen by users which usually have low entropy and weak randomness, and as a result cannot be used as cryptographic keys.

While cannot be used directly as cryptographic keys, the passwords can still be used to derive cryptographic keys, by using Key Derivation Function (KDF).

This document defines a new scheme, Key-Derivation Authentication Scheme, to replace the Digest scheme, that could be used with the challenge-response authentication framework used by SIP to authenticate the user.

The Key-Derivation scheme ensures that the password is never sent on the wire, and allows for a better secure storage of passwords, as it significantly increases the amount of computation needed to derive a key from a password in a dictionary attack.

The Key-Derivation scheme creates a master-key, that is derived from the password, which has a much better entropy than the password, to calculate a proof-of-possession for the shared password.

1.1 Terminology

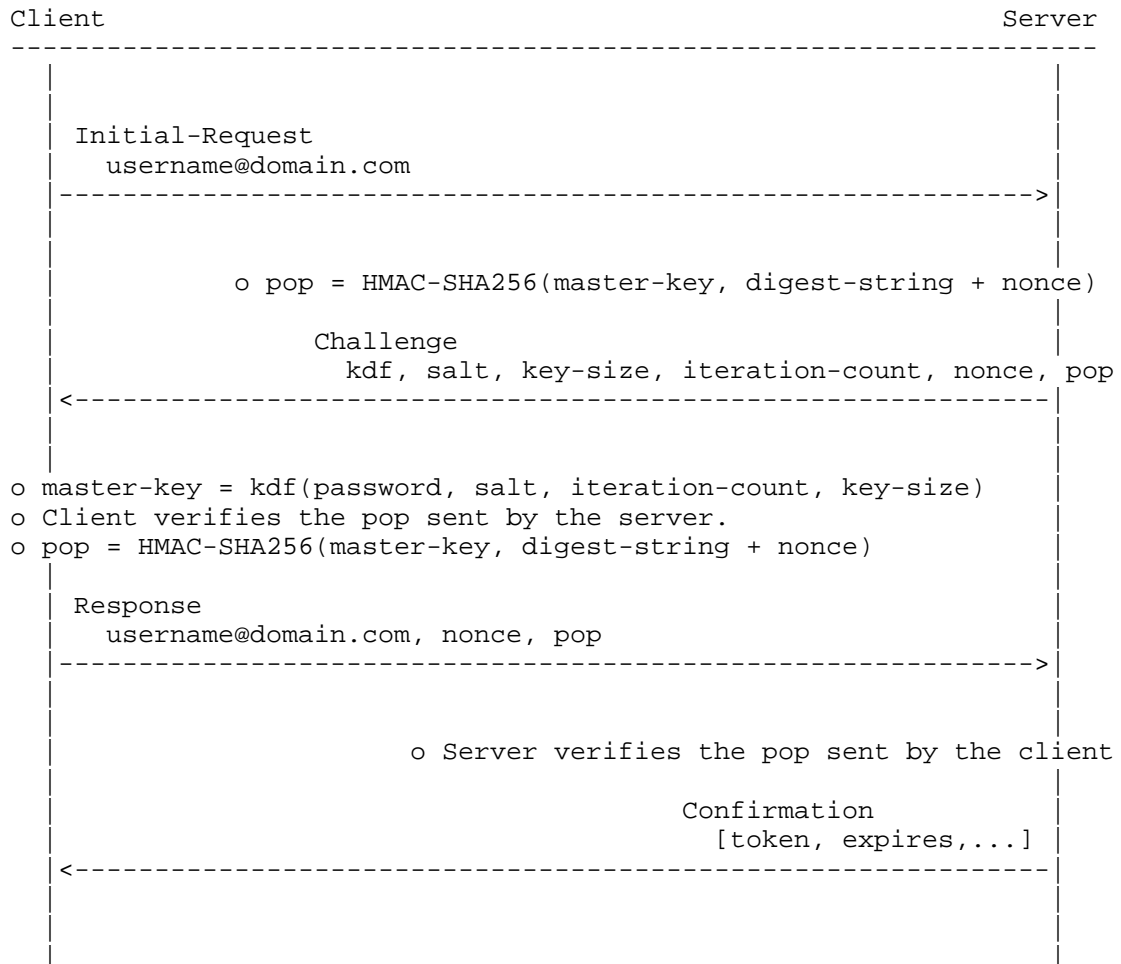
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2 Operations Overview

When an account is created, the server uses a KDF, a salt, a key length, and an iteration count to create a master-key based on the user's password, as defined in [NIST-KD]. The server then stores the following information in the database:

- o username
- o iteration count
- o salt
- o master-key

The following flow describes at a high-level the flow of messages based on the challenge-response framework:



With the challenge-response framework the initial request from the client is sent without providing any credentials.

When the server receives the initial request from the client, the server fetches the master-key associated with the username provided in the request. The server then uses the master-key to create a proof-of-possession (pop) using an HMAC-Hash function with the digest-string and nonce from the challenge.

The digest-string, as defined in Section 9 of [RFC4474], is a list of SIP headers that must be hashed to create the proof-of-possession defined in this document.

The server then challenges the request and includes the Key-Derivation scheme with a kdf, a salt, a key-size, an iteration-count, a nonce, and pop.

To be able to provide credentials to the server the client must create the master-key as was done by the server when the account was initially created as described above using the parameters provided by the server in the challenge. The client will then verify the pop sent by the server using its master-key, the digest-string of the incoming request, and the nonce provided in the challenge.

The client then creates a proof-of-possession (pop) using an HMAC-Hash function and the master-key using the digest-string from the response concatenated with the nonce to be sent to the server. A valid response from the client will contain the Key-Derivation scheme, a nonce, and the pop parameter.

When the server receives the response, it verifies the pop, and if that is valid, it sends a confirmation.

At the end of the above process, the client and the server would have established a communication channel after completing a mutual authentication using the same master-key on both sides.

Subsequent requests will be able to use the master-key to create pop to prove possession of the credentials.

3 The Challenge

When a server receives a request from a client, and an acceptable authorization is not sent, the server challenges the originator to provide credentials by rejecting the request and include the Key-Derivation scheme.

The challenge should include the following parameters:

KDF (REQUIRED)

A deterministic algorithm used to derive cryptographic keys from a shared secret like a password. A good example of such a function is HMAC-SHA2-256.

Iterations (OPTIONAL)

The number of iterations that the KDF will be applied on the salt and password. The default value for this parameter is 1000.

Salt (REQUIRED)

A random value that is used to make sure that the same password will always be hashed differently. The salt **MUST** be generated using an approved Random Number Generator.

Key-Size (REQUIRED)

The size of the derived key in bits.

nonce (REQUIRED)

A server-specified value that should be uniquely generated each time a challenge is made.

pop (REQUIRED)

The pop is derived from applying the HMAC-SHA256 on digest-string and a nonce using the master-key, as follows:

$$\text{pop} = \text{HMAC-SHA256}(\text{master-key}, \text{digest-string} + \text{nonce})$$

4 The Response

The client first creates the master-key based on the parameters provided by the server in the challenge.

The client then uses the master-key to verify the pop sent by the server; if that is successful, the client then uses the master-key to create a pop for the response to be sent to the server.

The client is expected to retry the request, passing the nonce and pop with the Key-Derivation scheme.

nonce (REQUIRED)

A client-specified value that should be uniquely generated each time a response is made.

pop (REQUIRED)

The pop is derived from applying the HMAC-SHA256 on digest-string and a nonce using the master-key, as follows:

$$\text{pop} = \text{HMAC-SHA256}(\text{master-key}, \text{digest-string} + \text{nonce})$$

5 The Confirmation

The server verifies the proof-of-possession sent by the client. If the verification is successful, the server sends a confirmation to the client; otherwise, the server declines the request.

6 Security Considerations

<Security considerations text>

7 IANA Considerations

<IANA considerations text>

8. Acknowledgments

<Acknowledgments text>

9 References

9.1 Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC4474] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.
- [NIST-KD] "NIST Special Publication 800-132 - Recommendations for Password-Based Key Derivations", December 2010.

<http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf>

9.2 Informative References

Authors' Addresses

Rifaat Shekh-Yusef
Avaya
250 Sydney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5267
Email: rifaat.ietf@gmail.com

SIPCORE Working Group
INTERNET-DRAFT
Intended Status: Standards Track
Expires: April 17, 2015

R. Shekh-Yusef, Ed.
Avaya
V. Pascual
Quobis
October 14, 2014

The Session Initiation Protocol (SIP) OAuth
draft-yusef-sipcore-sip-oauth-01

Abstract

This document defines an authorization framework for SIP that is based on the OAuth 2.0 framework, and adds a simple identity layer on top of that, based on the OpenID Connect Core 1.0, to enable Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License 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	Terminology	3
1.2	Definitions	4
1.3	Roles	4
1.4	ID Token	5
2	Benefits	5
2.1	Challenges	5
2.2	Single Sign-On	5
2.3	Level of Service	5
2.4	3rd Party Authorization	6
3	Authorization Code Grant type	6
3.1	Enterprise SSO Use Case	6
3.2	Justifications	6
3.2	Operations Overview	7
3.3	Registration	9
3.4	Authorization	10
3.5	Acquiring ID Token	11
3.6	Token Refresh	12
3.7	Authenticated Requests	12
3.8	Services	12
4	Resource Owner Password Credentials Grant type	13
4.1	SIP SSO	13
4.2	Operations Overview	13
4.3	Registration and Acquiring Tokens	15
4.4	Discarding Credentials	16
4.5	Token Refresh	16
4.6	Authenticated Requests	16
4.7	Examples	17
5	Client Credentials Grant	18
5.1	Registration	18
5.2	Authorization	19
6	Outbound	20
6.1	Authorization Code Grant type	20
6.2	Resource Owner Password Credentials Grant type	20

6.3	Client Credentials Grant type	20
7	Security Considerations	21
8	IANA Considerations	21
9	Acknowledgments	21
10	References	21
10.1	Normative References	21
10.2	Informative References	21
	Authors' Addresses	22

1 Introduction

The SIP protocol [RFC3261] uses the framework used by the HTTP protocol for authenticating users, which is a simple challenge-response authentication mechanism that allows a server to challenge a client request and allows a client to provide authentication information in response to that challenge.

The SIP protocol does not have an authorization framework to allow the system to control access to various services provided by the system.

OAuth 2.0 [RFC6749] defines a token based authorization framework to allow clients to access resources on behalf of their user. It also defines four types of authorization grants, which the client uses to request the access token.

The OpenID Connect 1.0 [OPENID] specifications defines a simple identity layer on top of the OAuth 2.0 protocol, which enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User.

This document defines an authorization framework for SIP that is based on the OAuth 2.0 framework, and adds the identity layer on top of that, based on the OpenID Connect Core 1.0 specification.

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 RFC 2119 [RFC2119].

1.2 Definitions

Types of SIP services:

- o Basic SIP Services: make/receive call, transfer, call forward, etc.
- o Advanced SIP Services: services provided by SIP application servers, e.g. Voice Mail, Conference Services, Video Services, Presence, IM, ...

Single Sign-On (SSO)

SSO is a property that allows the user to be authenticated once and as a result have access to multiple services in the system.

Authentication

The process of verifying the identity of a user trying to get access to some network services.

Authorization

The process of controlling a user access to network services and the level of service provided to the user.

1.3 Roles

resource owner

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

resource server

The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

OAuth 2.0 client

An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

SIP client

An application making requests to access SIP services on behalf of the end-user.

authorization server

The server issuing access tokens to the OAuth 2.0 client after successfully authenticating the resource owner and obtaining authorization, or the server issuing ID tokens to the SIP client after successfully authenticating the end-user.

proof-of-possession (pop)

A hash used by one party to prove to another party that it is in possession of some shared credentials, without sending the credentials on the wire.

1.4 ID Token

RFC6749 defines two types of tokens: access token and refresh token. This document defines a new token: ID Token as defined in [OPEN-ID].

ID tokens are credentials used by the SIP client to access SIP services on behalf of the end-user.

An ID token is a string representing an authorization issued to the SIP client. The string is usually opaque to the SIP client. Tokens represent specific scopes and durations of access, granted by the SIP system, and enforced by the SIP proxy, SIP application servers, and the authorization server.

2 Benefits

This section describes the benefit of this authorization framework:

2.1 Challenges

With the existing mechanism, the proxy and application servers might need to challenge many of the requests sent by a client, which adds traffic that could be avoided with this authorization mechanism.

2.2 Single Sign-On

Single Sign-On is a property that allows the user to be authenticated once and as a result have access to multiple services in the system.

This authorization mechanism would enable Single Sign-On, as the user will be authenticated once and as a result given a token and a refresh token to allow the user access to various services based on the token scope.

2.3 Level of Service

This authorization mechanism allows the application server to control the level of service provided to the user based on the token scope.

2.4 3rd Party Authorization

This authorization mechanism allows the user to be authenticated and obtain tokens using some 3rd Party Authorization mechanism and still get services from the system.

3 Authorization Code Grant type

3.1 Enterprise SSO Use Case

An enterprise is interested in providing its users with an SSO capability to the corporate various services. The enterprise has an authorization server for controlling the user access to their network and would like to extend that existing authorization server to control the user access to the various services provided by their SIP network.

The user is expected to provide his corporate credentials to login to the corporate network and get different types of services, regardless of the protocol used to provide the service, and without the need to create different accounts for these different types of services.

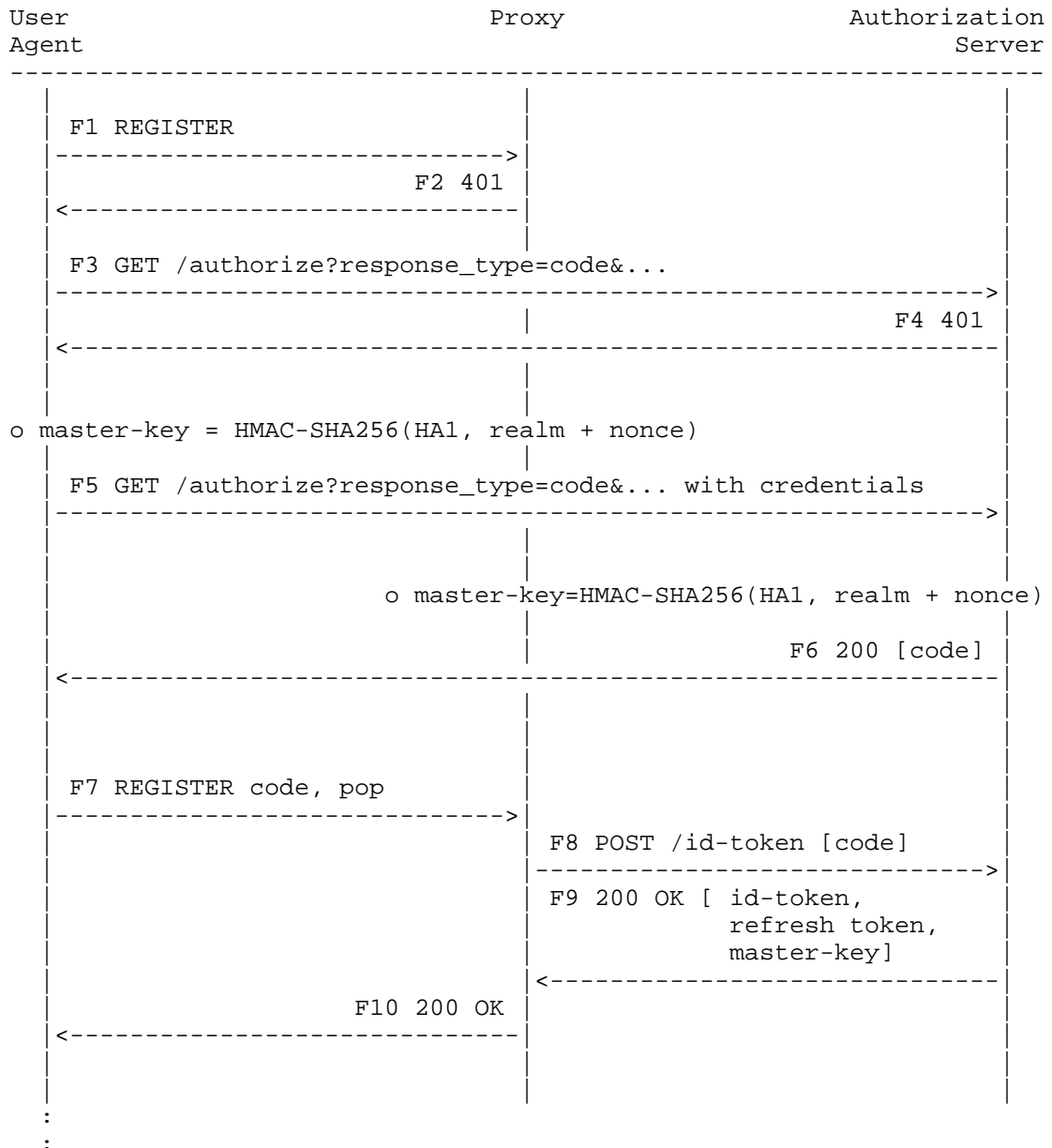
3.2 Justifications

There are 3 reasons that justify the use of the authorization code grant type:

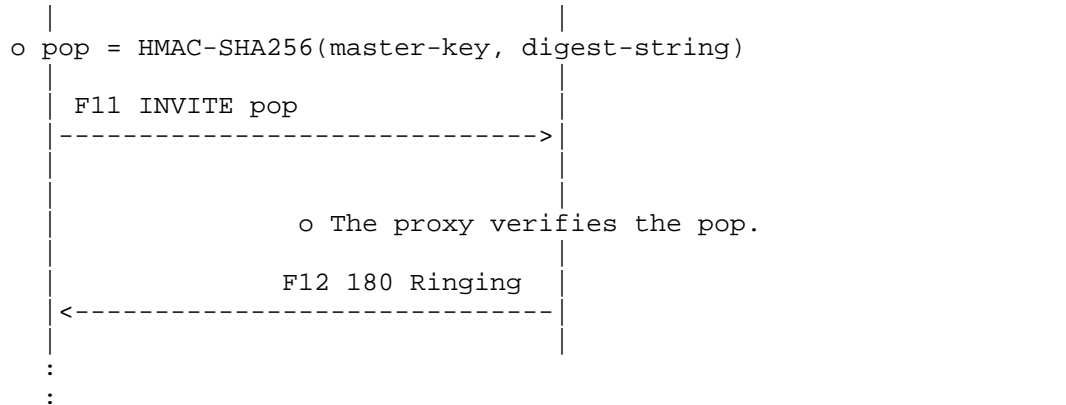
1. Minimize the potential for exposing the token.
2. Enable the proof-of-possession mechanism.
3. Re-use of existing authorization server that already supports this grant type.

3.2 Operations Overview

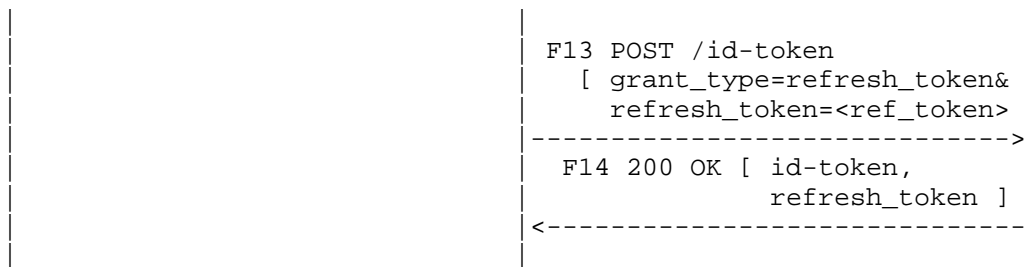
The following figure provides a high level view of flow of messages for the Authorization Code Grant type:



Subsequent Requests



Token Refresh



During registration, if the UA is in possession of a valid ID Token, the UA could use the token to register with the proxy; otherwise, the UA initially sends a REGISTER request (F1) without providing any credentials.

The proxy challenges the UA by responding with 401 (F2) that includes the address of the Authorization Server.

[[OPEN ISSUE]]

How should the UA be redirected to the Authorization Server:

1. New SIP parameter?
2. Extend the Bearer scheme?
3. Define a new Scheme?

The UA will then contact the Authorization Server without providing any credentials in the first request (F3). The Authorization Server challenges the request using the Digest scheme (F4), and the client retries the request (F5) and provide the user's credentials.

The Authorization Server verifies the request from the client; if the verification is successful, the Authorization Server responds with 200 OK (F6) includes a code in the body part.

The UA then retries the request (F7) and include the code in the body of the request. The proxy then contacts the Authorization Server and exchanges the code for a token (F8 & F9).

3.3 Registration

The UA initiates the process by sending a REGISTER request (F1) to the proxy. The proxy will redirect the UA to the Authorization Server by responding with 401 (F2) that include the address of the Authorization Server in the form of an HTTP URI.

The UA will then follow the authorization steps defined in section 3.4. At the end of the authorization process the UA will have a code that it will use to complete the registration process.

The UA will send a new REGISTER request (F7) and include the code in the body of the request with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The proxy will then use the code to get a token from the Authorization Server as defined in section 3.5. If the proxy is able to obtain the token, the proxy will respond with 200 OK (F10) to the UA to complete the registration process.

3.4 Authorization

The UA constructs the initial request (F3) without providing any user credentials, but with the following URI parameters in the query component:

response_type (REQUIRED)

Value MUST be set to "code".

user_id (REQUIRED)

The user's address-of-record (AOR).

scope (OPTIONAL)

The scope of the access request as described by Section x.x.

state (RECOMMENDED)

The value of this parameter is a nonce created by the client to prevent replay attack. The nonce is a uniquely generated value for each request. This parameter might not be included with the initial request that does not include credentials (F3).

The Authorization Server uses the user's AOR specified in the user_id parameter to verify that the user has an account in the system, and then challenges the request by responding with 401 (F4) with Digest scheme.

The UA will generate a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user's HA1 and the concatenation of realm and nonce received in the challenge from the server.

The UA will then send a new authorization request (F5), but this time include the credentials requested by the server. The UA will use the same parameters values used in the initial authorization request with the exception of the state parameter which will get a new nonce value.

When the server receives the request with the credentials (F5), the server will verify the digest provided by the UA; if that is successful, the server will respond with 302 (F6) and include a code in the body of the response with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The server then generates a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user's HA1, and the concatenation of realm and nonce sent in the challenge (F4) to the client.

3.5 Acquiring ID Token

The proxy receives the REGISTER request (F7) that includes a body with a code obtained during authorization (section 3.4). The proxy will then contact the Authorization Server to exchange the code with an ID Token.

The proxy sends a POST request (F8) to the Authorization Server and include the following parameters in the body:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

If the request is valid and authorized, the authorization server responds with a 200 OK (F9) to complete the registration process, with id_token, token_refresh, and the master-key in the body.

3.6 Token Refresh

The proxy makes a refresh request to the token by sending a refresh POST request (F13) that includes a body with the `grant_type` and the `refresh_token`.

For example:

```
grant_type=refresh_token&refresh_token=<refresh_token>
```

If the proxy fails to refresh the token, then it MUST challenge the next request from the UA, and as a result the UA MUST go through the authorization process defined in section 3.4 to obtain new tokens.

3.7 Authenticated Requests

When the UA wants to send any request to the proxy, it MUST include the Authorization header and use the Bearer scheme to carry the proof-of-possession of the master-key.

The pop is calculated using the master-key as follows:

```
pop = HMAC-SHA256(master-key, digest-string)
```

The following is an example of an Authorization header with Bearer scheme:

```
Authorization: Bearer pop=<pop>
```

See rfc4474, section 9, for the SIP headers to hash to create digest-string.

[[OPEN ISSUE]] The Bearer scheme is used to deliver tokens without providing any proof of possession. We probably need to use different scheme later on.

3.8 Services

When the UA tries to access a service on behalf of a user, e.g. Voice Mail Service, the proxy forwards the request to the server providing the service and MUST include an Authorization header with the Bearer scheme that carries the token needed to get service, as follows:

```
Authorization: Bearer token=<token>
```

4 Resource Owner Password Credentials Grant type

4.1 SIP SSO

An enterprise is interested in providing its users with an SSO capability to the corporate various SIP services.

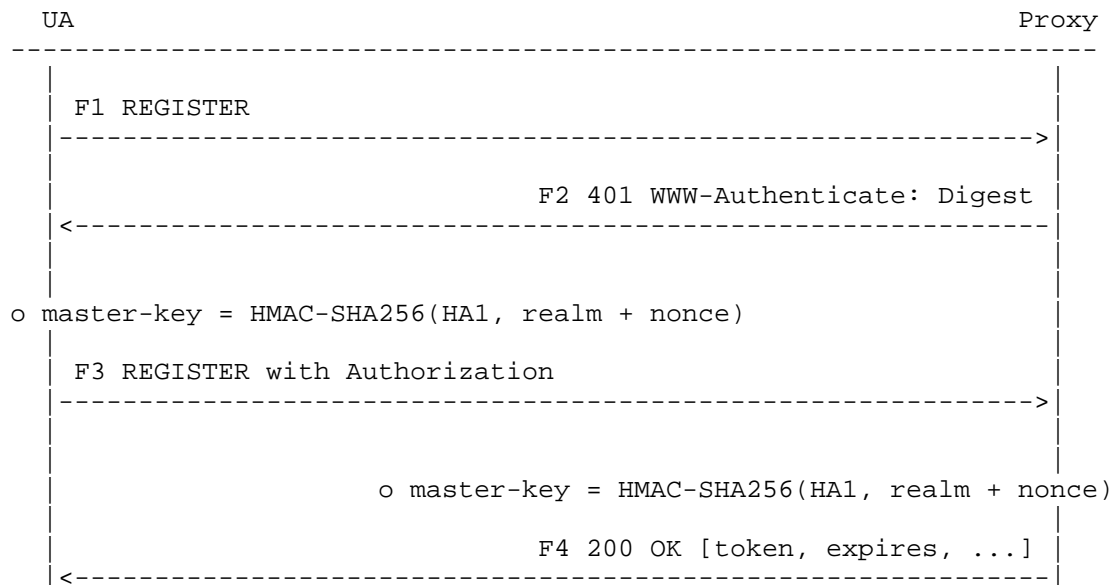
The enterprise wants to control the services provided to their SIP users and the level of service provided to the user by their SIP application servers without the need to create different accounts for these services.

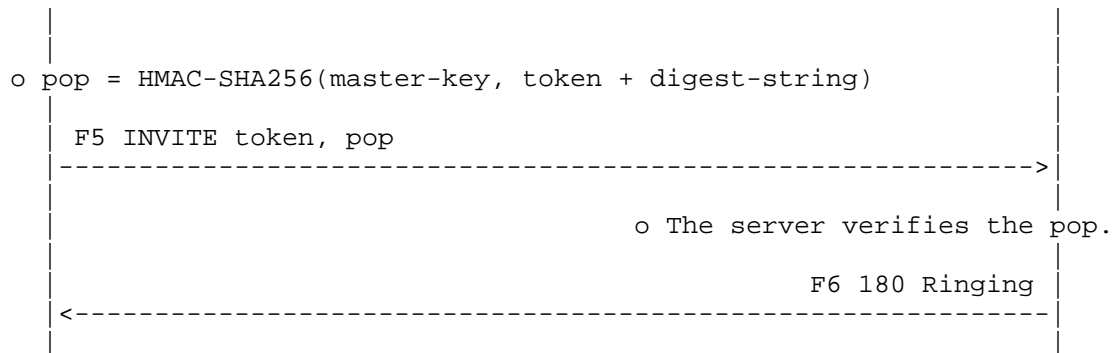
The enterprise wants to utilize an existing authentication mechanism provided by SIP, but would like to be able to control who gets access to what service and when.

The user is expected to use his SIP credentials to login to the SIP network and get access to the basic services, and to get access to the services provided by the various SIP application servers without being challenged to provide credentials for each type of service.

4.2 Operations Overview

The following figure provides a high level view of flow of messages for the Resource Owner Password Credentials Grant type:





During registration the UA initially sends a REGISTER request (F1) without providing any credentials.

The proxy then challenges the UA by responding with 401 (F2) that includes the Digest scheme in the www-authenticate header.

The UA will generate a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user's HA1 and the concatenation of realm and nonce received in the challenge from the server. The UA will continue to use the existing operation of handling the Digest challenge and then sends a new REGISTER request (F3) with the credentials to the server.

When the server receives the request with the credentials (F3), the server will verify the digest provided by the UA; if that is successful, the server will accept the registration (F4) and include the details of the token in the response.

The server then generates a master-key that is based on an HMAC-Hash algorithm, e.g. HMAC-SHA256, that takes an input the user's HA1, and the concatenation of realm and nonce sent in the challenge to the client.

At the end of the above process the UA would have registered with the proxy and both the UA and the proxy would have created the same master-key without sending the master-key on the wire.

Later when the UA wants to send a request to the proxy it MUST always include the token and SHOULD include the pop as defined in section 4.6.

4.3 Registration and Acquiring Tokens

The UA MUST request the access token during the registration process with the proxy, by including a body with the grant_type as "password". Initially, the UA sends a REGISTER request without providing any credentials.

The proxy MUST then challenge the UA by responding with 401 with the Digest scheme in the WWW-Authenticate header.

When the UA gets challenged by the proxy to provide its credentials, the UA MUST include its credentials in the new REGISTER request in the authorization header as it is done with the existing mechanism, and MUST include a body with the grant_type as "password".

In addition, the UA MUST generate a master-key as follows:

master-key = HMAC-SHA256(HA1, realm + nonce)

- o HA1 - this is the user's H(A1) as defined in [DIGEST].
- o realm - this is the realm that is returned by the server in the response to the initial request from the UA.
- o nonce - this is the nonce that is returned by the server in the response to the initial request from the UA.

When the server receives the request with the credentials, the server will verify the digest provided by the UA; if that is successful, the server will accept the registration and include the details of the token in the response.

[[OPEN ISSUE]]

How should the tokens be transported to the UA? in the body of the 200 OK? or a SIP header?

The server then generates a master-key following the same procedure followed by the client.

As a result of this procedure both the UA and the server would have created the same master-key without sending the master-key on the wire.

4.4 Discarding Credentials

After successfully receiving the access and refresh tokens from the proxy, the UA SHOULD discard the user credentials.

4.5 Token Refresh

The UA makes a refresh request to the token by sending a refresh REGISTER request that includes the authorization header and a body with the grant_type, the refresh_token, and the proof-of-possession of the master-key.

For example:

```
grant_type=refresh_token&refresh_token=<refresh_token>&pop=<pop>
```

4.6 Authenticated Requests

When the UA wants to send any request to the proxy, it MUST include the Authorization header and use the Bearer scheme to carry the access token, and the proof-of-possession of the master-key. For example:

```
Authorization: Bearer token=<token>, pop=<pop>
```

See rfc4474, section 9, for the SIP headers to hash to create the value for the proof.

[[OPEN ISSUE]]

The Bearer scheme is used to deliver tokens without providing any proof of possession. We probably need to use different scheme later on.

4.7 Examples

```
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/TCP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 19
```

```
grant_type=password&pop=<pop>
```

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
;received=192.0.2.4
To: Bob <sip:bob@biloxi.com>;tag=2493k59kd
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

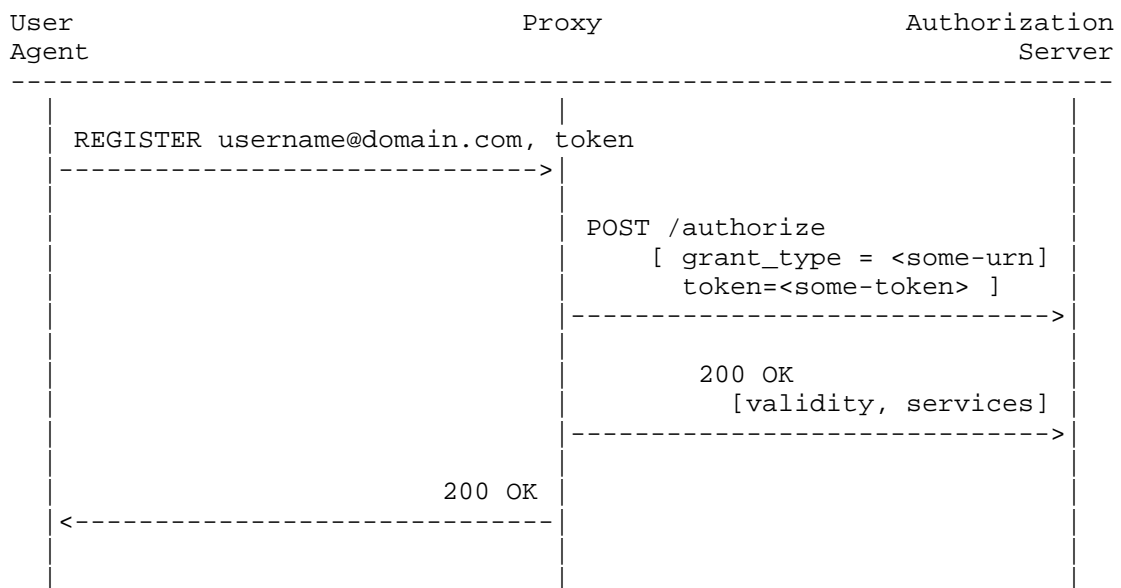
```
{
  "access_token": "2YotnFZFEjrlzCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter": "example_value"
}
```

5 Client Credentials Grant

The following flow assumes that the UA is able to get a token using some out-of-band mechanism, and the UA wants to use the token to register, subscribe, and get service.

The flow uses a combination of the following from RFC6749:

- o Client Credentials Grant defined in section 4.4
- o Extensions Grants defined in section 4.5.



5.1 Registration

The UA is in possession of a token that was obtained through some out-of-band mechanism.

The UA sends a REGISTER request and include the token in the Authorization header using the Bearer scheme as defined in RFC6750.

If the proxy is able to verify the token, the proxy accepts the registration request and responds with 200 OK.

5.2 Authorization

When the proxy receives the REGISTER request with the token, the proxy will try to first validate the token before responding to the UA request.

The proxy sends a POST request and include the following parameters in the body of the request:

grant_type (REQUIRED)

Some well defined URN.

username (REQUIRED)

The resource owner username.

access_token (REQUIRED)

The token received from the UA.

scope (OPTIONAL)

The scope of the token.

If the authorization server is able to validate and authorize the request, it will respond with 200 OK with a body that contains the following parameters:

access_token, token_type, expires, refresh_token, scope

6 Outbound

RFC5626 defines a mechanism that allows a UA to simultaneously connect and establish registration with multiple outbound proxies to get service.

This section describes that impact of outbound on this authorization mechanism.

6.1 Authorization Code Grant type

During initial registration with the primary proxy, the UA is able to get an authorization code that it will use to register with the primary proxy. Assuming the authorization server is shared between the various outbound proxies, the UA will be able to use the same authorization code to register with the secondary proxies and as a result each one of the secondary proxies will get the master-key associated with the user to be used for the calculation of the proof-of-possession.

6.2 Resource Owner Password Credentials Grant type

During registration the proxy challenges the UA, and both the proxy and the UA create a master-key based on HA1, realm, and nonce. Since the nonce is not shared between the various proxies, it is not possible for the outbound proxies to use the same master-key; as a result, the UA is expected to maintain a master-key and token per outbound proxy.

6.3 Client Credentials Grant type

Since the tokens are obtained using some out-of-band mechanism, and the authorization server is shared between the outbound proxies, the UA should be able to register and get service from any one of the outbound proxies.

7 Security Considerations

<Security considerations text>

8 IANA Considerations

<IANA considerations text>

9 Acknowledgments

<Acknowledgments text>

10 References

10.1 Normative References

- [KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.
- [RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.
- [OPENID] Sakimura, N., J. Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., "OpenID Connect Core 1.0", February, 2014 http://openid.net/specs/openid-connect-core-1_0.html
- [DIGEST] Shekh-Yusef, R., Ahrens, D., and Bremer, S., "HTTP Digest Access Authentication", Work in Progress, January 2014.
- <https://datatracker.ietf.org/doc/draft-ietf-httpauth-digest/>

10.2 Informative References

Authors' Addresses

Rifaat Shekh-Yusef (Editor)
Avaya
250 Sydney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5267
Email: rifaat.ietf@gmail.com

Victor Pascual
Quobis
Spain

Email: victor.pascual@quobis.com