SIP F. Cao
Internet-Draft C. Jennings
Expires: January 11, 2006 Cisco Systems
July 10, 2005

Response Identity and Authentication in Session Initiation Protocol
draft-cao-sip-response-auth-00

Status of this Memo

Copyright Notice

Abstract

This draft describes some extensions for verifying SIP response identity and enhancing SIP response authentication. Some mechanisms are demonstrated for providing and verifying the identity of SIP responses. In order to prevent several kinds of security attacks through SIP response, SIP response authentication should be provided through a chain of trust of the SIP responses. Some extensions are proposed to enhance the per-hop authentication for handling SIP response.

This draft is an early work in progress and suggests some approaches but there is still significant discussion needed. Some of the attacks discussed in this draft can be mitigated by using the sips URL.

Table of Contents

1. Introduction

This document provides enhancements for addressing security concerns on response messages in Session Initiation Protocol (SIP [1]). There are some limitations with the current handling of SIP response without identity verification and authentication that leaves holes for malicious attacks through SIP response.

[3] described the current limitations of some security mechanisms provided in SIP [1]. Due to these limitations, some extensions were added in [3] to address the need for authenticating identity of SIP request.

The identity of SIP response is more complicated than that of SIP request. First, SIP response may be originated by any intermediate SIP proxies instead of the desired SIP UAS. Because SIP UAC may send requests to SIP UAS without any previous association, these intermediate SIP proxies may not be known or verified by SIP UAC beforehand. Second, the presence of the exact responder for SIP response is not clearly defined, which is different from the From header field for SIP request. In general, it is obvious that the To header field cannot be used as described above. Contact and Reply-to have their own meanings and cannot be relied on for backward compatibility.

In this document, some mechanisms are demonstrated to enable the sender to verify the identity of a corresponding SIP response.

Furthermore, there are still some loopholes left for malicious attacks through SIP responses. In particular, there is no strict per-hop authentication for the received SIP response. This could enable an attacker to spoof SIP response and disturb the SIP service.

This issue is defined as Chain of SIP Response Trust (CSRT) in this document. Some extensions are shown in this document to enhance CSRT in SIP.

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 [2].

Domain-based Authentication Service (DAS): Authentication service is provided for each domain through its certificate and the domain private key. Proxies may authenticate servers with the domain keys.

Authenticated Identity Body (AIB): some SIP headers are replicated

into an S/MIME body of the same message and are signed with a digital signature (See [5])

Chain of SIP Response Trust (CSRT): as described in Section 1.

Certificate: An X.509v3 [15] style certificate containing a public key and a list of identities in the SubjectAltName that are bound to this key. The certificates discussed in this document are generally self signed and use the mechanisms in the SIP Identity specification to vouch for their validity.

3. Overview

This section gives an overview of the requirements and the mechanisms for addressing the security concerns of SIP response. In particular, the first part is about SIP response identity and how to verify it. The rest is about CSRT for guaranteeing per-hop authentication to prevent malicious attacks through SIP response.

3.1 SIP Response Identity

SIP response identify is crucial for negotiation and providing the desired services. UAC might guess the identity of the responder of the received SIP response message through the response code and some header fields. But there is no defined mechanism for determining that identity and verifying it.

The following requirements should be addressed:

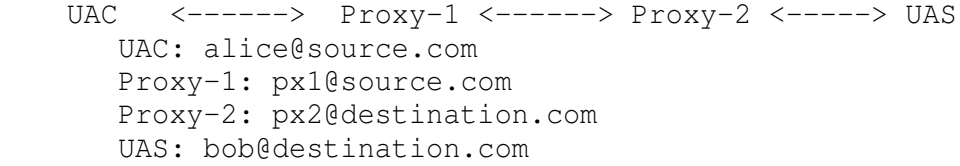
O The identities of both UAs and proxies should be covered

O The mechanism should be backward compatible.

O The identity should be clearly specified for the responder of the SIP response message.

O The integrity of SIP response should be covered along with the responder identify

The following example is used in this document to demonstrate the mechanisms in many sections:



Alice sends an INVITE request to Bob. Proxy-2 receives the request and informs Alice of the response code 183 Session Progress, along with two new header fields called Responder and Responder-Info:

```
Responder: claimer=px2@destination.com;
           verify-method=DAS;
Responder-Info: https://www.destination.com/certs
Identify: akfjiqiowrgnavnvnnfa2o3fafanfkfjakfjalkf203urjafskjfaf
          Jprqiyupirequqpiruskfka

 [Identity needs to be recalculated]
```

The field of claimer specifies the exact identity of the responder. The field of verify-method indicates the secure mechanism for verifying the identify of the responder.

There are several security methods covered in this document to support this mechanism:

O DAS

O AIB (See Appendix A)

For DAS, the mechanism is similar to [3]. Some headers, including the new header Responder, and the body of the message are used to compute a hash. This hash is signed with certificate for Proxy-2's domain (destination.com), and the final output is inserted into the header field Identity introduced by [3]. One new header, Responder, is introduced to specify the exact responder and related authentication method. Responder-Info is inserted to indicate where to acquire the certificate for the claimer of the responder.

For DAS, the proxy servers can obtain the certificate of DAS for the responder through Responder-Info. The digest in Identity can be verified for the responder identity. If there is a mismatch, the proxy server may replace the response code with 431 Failed Responder Identity for indicating the problem as early as possible.

3.2 Chain of SIP Response Trust

In order to prevent several kinds of malicious attacks through SIP response, Chain of SIP Response Trust (CSRT) should be provided to enhance the per-hop authentication for receiving SIP response.

For example, in the above example, a rogue proxy can spoof the IP address of Proxy-2 and send the response back to Proxy-1 along with its rogue domain authentication service info, before Proxy-2's

response. Without the per-hop authentication, Proxy-1 will be deceived by the response from the rogue proxy.

The following requirements should be addressed:

O authentication between neighboring domains or nodes can be enhanced

O The mechanism should be simple

O CSRT can be built when this mechanism is applied on all the hops.

One simple authentication mechanism is proposed in this document for satisfying all these requirements. This mechanism is to generate a digest challenge for the next-hop node (or domain). The authorization to this challenge should be delayed and piggybacked with the next normal SIP response from the next-hop downstream node (or domain). After the digest is verified, the trust can be enhanced for the SIP response from the next-hop node (or domain).

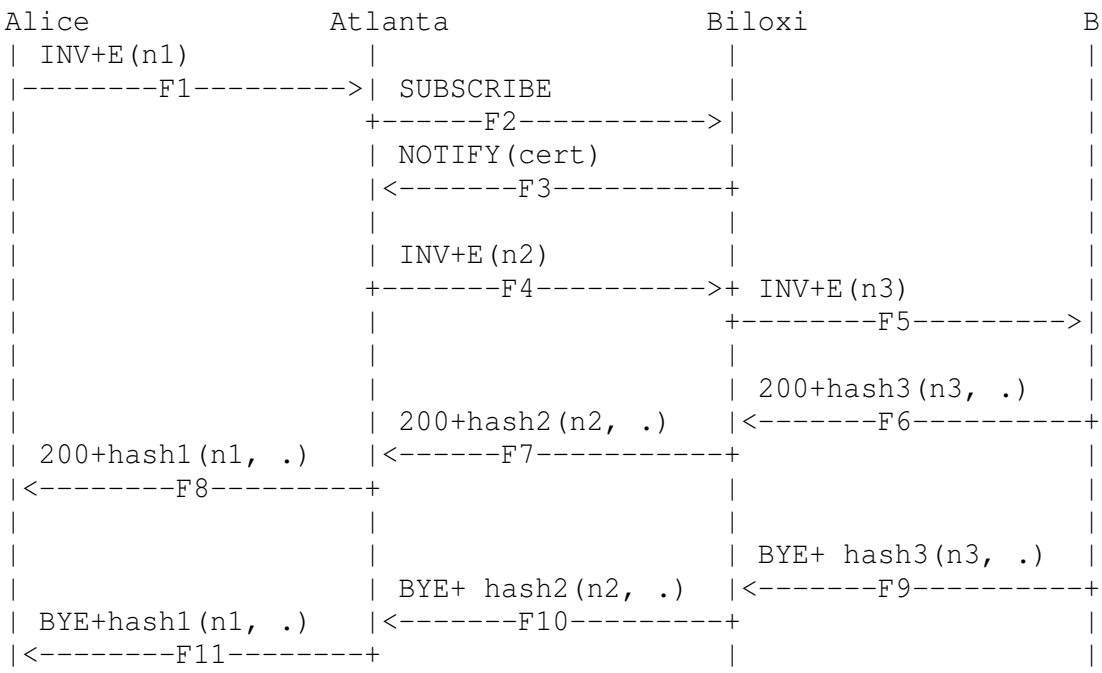
There are several security mechanisms covered in this document to support this mechanism:

O DAS

O shared secret key with the next-hop downstream node

O public key of the next-hop downstream node

The figure below shows a basic call to illustrate some scenarios. The call is initiated by alice@atlanta.com to bob@biloxy.com. The assumption is that Alice and Atlanta have a shared secret, Biloxi has a public certificate, and Bob and Biloxi have a shared secret.



In message F1, Alice sends a normal invite but includes an Authentication header that includes the encrypted nonce, n1, that is encrypted for the next hop, which is Atlanta.

In message F4, Atlanta will forward the invite to Biloxi with a nonce that is encrypted for Biloxi; however, to do the encryption, Atlanta may have to use the SUB/NOT in message F2 and F3 to fetch Biloxi's public key so that Atlanta can encrypt the nonce. Note F2 and F3 might have already been done for previous SIP dialogs from Atlanta.com to Biloxi.com.

In message F5, biloxi sends the INVITE with a nonce encrypted for Bob, using the shared secret between Biloxi and Bob.

In message F6, Bob inserts a header that says the responder in bob@biloxi.com and computes a hash over key parts of the message including the responder header field value. The hash includes the decrypted content of the nonce that Biloxi sent to Bob. When Biloxi receives this message it can verify that the hash is correct and that it believes the responder information.

Biloxi computes a new hash over the message using the nonce2 and sends F7 using this hash.

Later in message F9, F10, and F11, the hash can be computed using the previous nonces. The proxies do not need to be session state-full, as long as the nonce are constructed such that the proxy can later

check that they are only being used in the dialog for which they were originally constructed.

If the verification in Biloxy or Atlanta indicates the unmatched SIP response authorization, the proxy may replace the response code with 432 Failed Response Authorization for announcing the failure of the next-hop response authentication.

There are some advantages of this mechanism. For example, man-in-the-middle attacks can be prevented as the rogue proxy does not have the message forward to him in a valid way and cannot compute a valid hash for the response. This method can be easily distributed to enhance the security in any specified hops among domains.

A proxy such as Biloxy does not need to do work until Bob actually sends the 180 response. At this point it must decrypt the the original nonce and recompute the hash. However, this is after the call has been at some level accepted by a device that this provides service for.

Therefore, CSRT can be enhanced through this extension from end to end. The rogue proxies can be prevented from attacking SIP services through SIP responses.

4. User Agent Behavior

The extensions in this document require new processing and parsing for both UAS and UAC. Their behaviors are described in this section.

4.1 SIP Response Identity

When UAS sends the response, UAS must accurately generate the new header fields as the responder.

For DAS, UAS must populate Responder inside the SIP response. In addition, the URI as claimer inside Responder must be consistent with what UAS registers in its domain. Note the URI as claimer may be different from other header fields, such as Reply-To, Contact, and To, in some scenarios. Please see Proxy Server Behavior for Identity and Responder-Info.

When it receives the corresponding SIP response, UAC can verify the identify of the responder. For DAS, the certificate of DAS for the responder should be obtained to verify the digest in Identity.

UAC may receive the response code 431 Failed Responder Identity. UAC should choose to avoid the verification of the responder identity. UAC should treat it as a failure and may terminate the dialog.

4.2 Chain of SIP Response Trust

When UAC sends the SIP request, UAC can generate nonce before assembling the new authentication header field.

For DAS, UAC must obtain the certificate of DAS for the next-hop node. The nonce is encrypted and inserted into Response-Authentication. For the shared key with the next-hop node, the nonce is encrypted by the shared key to ensure its privacy.

When it receives the SIP response for the corresponding SIP request, UAC should verify the authorization from the next hop. It generates its own digest through its saved nonce in decrypted format, plus some header fields and the message body in response message. This digest is compared with the one in SIP response message from the next hop. If there is a mismatch, it should treat it as an error and may terminate the dialog with the failure reason.

Even if UAC may receive the response code 432 Failed Response Authorization, UAC should finish the steps for verifying the received response from the next-hop. If Response-Authorization carries the correct digest, this response code can be trusted. The proper follow-up operations should take place, such as terminating the dialog with the failure reason. If not, the received response may be suspicious. UAC should analyze the reason before taking any steps for further operations.

As a recipient of the SIP request with Response-Authentication, UAS should generate the digest for SIP response with respect to the specified method. The digest is inserted into UAS's next SIP response.

5. Proxy Server Behavior

The extensions in this document require new processing and parsing for proxy servers. Their behaviors are described in this section.

5.1 SIP Response Identity

The proxy server may provide the domain authentication service for an outgoing SIP response. When a SIP response is received without the header Responder, the proxy server may insert the identity of the sender as the responder along with Responder-Info and Identity.

After receiving the SIP response with a new header field Responder, the proxy servers may verify the responder identity in order to detect the mismatched identity as early as possible.

For DAS, the proxy server can obtain the certificate of DAS for the responder through Responder-Info. The digest in Identity can be verified for the responder identity.

If there is a mismatch, the proxy server may replace the response code with 431 Failed Responder Identity for announcing the problem. On the other hand, the proxy servers may relay the SIP responses without checking the responder identity and modifying any fields including response codes.

5.2 Chain of SIP Response Trust

After receiving the SIP request with Response-Authentication, the proxy server must save the nonce received from the upstream node.

It is recommended that when the proxy server relays the SIP request, the proxy server carry its own Response-Authentication inside the request. The nonce should be encrypted.

Before relaying the SIP request to the next-hop downstream node, the proxy server should generate its own nonce, encrypt the nonce, and overwrite the Response-Authentication header field inside the SIP request.

For DAS, the nonce is encrypted by the certificate of the next-hop domain and inserted into Response-Authentication. For the shared key with the downstream node, the nonce is encrypted by the shared key to ensure its privacy.

Note that to reduce the risk of disclosure, the nonce received from the previous hop should not be forwarded to the next hop.

If the SIP response is received, the proxy server must finish two steps. First, it has to verify the authorization from the next-hop downstream node. It generates its own digest through its saved nonce in decrypted format, plus some header fields and the message body in response message. This digest is compared with the one in the SIP response message from the next hop.

Second, the proxy server has to generate another digest from the decrypted nonce received from the upstream node, some header fields, and the message body for SIP response. This digest is inserted into its relayed SIP response to the upstream node.

Note that the proxy server has to obtain the certificate, the public key or the shared key with the downstream node (or domain) before Response-Authentication is assembled. [4] is recommended to retrieve the certificate through SUBSCRIBE and NOTIFY in the enhanced

certificate management.

When it receives the SIP response for the corresponding SIP request, the proxy server should compare the digest inside Response-Authorization with its generated one. If there is a mismatch, the proxy server should analyze this suspicious response. The proper follow-up operations should take place, such as replacing the response code with 432 Failed Response Authorization. Note that the saved digest for the corresponding SIP request should be piggybacked into its response.

Even if it receives the response code 432 Failed Response-Authorization, the proxy server should finish the steps for verifying the validness of this received response from the downstream node.

6. Syntax and Examples

6.1 Header Syntax

Four new SIP headers are introduced in this document. Responder, Responder-Info, and Response-Authorization appear in the response. Response-Authentication is eligible in the request.

```
Responder = "Responder" HCOLON responder-param
Responder-param = claimer-param *( SEMI verify-param)
claimer-param = "claimer" EQUAL (name-addr / addr-spec)
verify-param = "verify-method" EQUAL ("DAS" / token)
```

Note: token in verify-param can be extended to cover other verification methods, such as AIB(See Appendix A in detail).

```
Responder-Info = "Responder-Info" HCOLON responder-info-param
responder-info-param = LAQUOT absoluteURI RAQUOT
```

For DAS, the responder's identity is the digest in the the Identity header. This digest is generated by including the following elements of the SIP response in a bit-exact string in this specified order.

- addr-spec in To
- addr-spec in From
- addr-spec of claimer field in Responder
- callid from Call-ID

O the digits and the method from CSeq

O Date field

O body content of the message with the bits exactly as they are in the message (in the ABNF for SIP, the message body).

In summary, digest-string for Identity header in the SIP response is

```
digest-string = addr-spec ":" addr-spec ":"
                addr-spec ":" callid ":" 1*DIGIT SP method
                ":" SIP-Date ":" message-body
```

Similar to [3], this digest-string is hashed and signed with the certificate for the domain. The mandatory procedure is sha1WithRSAEncryption as described in RFC 3371 with base64 encoding as described in RFC 3548.

Here is one sample response from Bob in the above example:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP px1.source.com;branch=z9hG4bKnashds8
 ;received=101.37.45.98
Via: SIP/2.0/UDP px2.destination.com;branch=bfajk34lk2
 ;received=121.56.12.1
To: Bob <sip:bob@destination.com>;tag=a6c85cf634
From: Alice <sip:alice@source.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Responder: claimer=bob@destination.com; verify-method=DAS
Responder-Info: https://www.destination.com/certificate
Identity: oiurw20984oij12kfqfknrewqfhgahg198431ufadsafafdag32r4189f
hafaaafi298r3398i32uip293gDFQqireu904328FQWlkafqroiewrjafaf
k189ahffahjf4289981
Content-Length: 0
```

[*Identity: needs to be recalculated]

Two new headers are introduced for CSRT:

```
Response-Authentication = "Response-Authentication"
                          HCOLON resp-authen-param
resp-authen-param = auth-method-param * (SEMI nonce-param)
auth-method-param = "method" EQUAL auth-method-enum
auth-method-eum   = "DAS" / "SharedKey" / "PublicKey"
nonce-param       = "nonce" EQUAL "nonce-value"

Response-Authorization = "digest" EQUAL resp-author-digest
Resp-author-digest = LDQUOT 32LHEX RDQUOT
```

For the digest generated in Response-Authorization, the digest-string includes

- status code of the response
- addr-spec in To
- addr-spec in From
- addr-spec of claimer field in Responder
- method and nonce in Response-Authentication
- callid from Call-ID
- the digits and the method from CSeq
- Date field
- body content of the message with the bits exactly as they are in the message (in the ABNF for SIP, the message body).

In summary, digest-string for Identity header in the SIP response is

```
digest-string = status-code ":"
                addr-spec ":" addr-spec ":" addr-spec  ":"
                auth-method-enum nonce-value ":"
                callid ":" 1*DIGIT SP method ":" SIP-Date ":"
                message-body
```

The decrypted nonce plus this digest-string are hashed and signed with the key based on the specified method. The mandatory procedure is sha1WithRSAEncryption as described in RFC 3371 with base64 encoding as described in RFC 3548.

7. Security Considerations

This document provides some security enhancements on SIP response identity and response authentication.

There are some advantages for the proposed mechanisms in this document. The new fields inside SIP response provide the needed responder identity with authentication methods, and are backward compatible with [1]. The mechanisms proposed for per-hop SIP response authentication can be easily used on any hops, such as hops between different domains, to prevent malicious attacks through SIP responses over those hops. Furthermore, if each hop (or all the hops with security concerns) is enhanced with these mechanisms, CRST can be created to detect and prevent several kinds of malicious attacks through SIP responses, and to guarantee the validness of SIP response.

For example, if a rogue proxy can sniff the SIP requests from Proxy-1 to Proxy-2, it can spoof the addresses and URIs of Proxy-2 and send the response back to Proxy-1 along with its own rogue domain authentication service info, before Proxy-2's response. Without the proposed mechanisms, Proxy-1 and the caller of SIP requests will be deceived by the response from the rogue proxy. This will allow the rogue proxy to conduct attacks, such as redirecting the requests to attack other targets for DoS attacks, redirecting the requests to rogue users for information disclosure, and terminating the dialogs for turning down SIP services.

With the mechanisms introduced in the document, Proxy-1 can detect the faked responses from the rogue proxy by checking the digest in Response-Authorization. These faked responses are dropped immediately by Proxy-1 without any impact on the callers of SIP requests.

Another example is to verify the response identity, which is important in many scenarios. This document provides the responder identity through the new header fields in SIP response, and the mechanism for verifying this identity.

All the hops with security concerns should apply these mechanisms for enhancing authentication for SIP response. If not, man-in-the-middle attacks may be possible again through SIP response, just as before.

This document is based on some existing results for domain-based authentication and certificate management (See [3, 4]). Therefore, these mechanisms may be affected by the secure concerns for these functional components.

As anonymous identity is a subject for future work, this document leaves one open question about the exact impact of these mechanisms on anonymous identity.

8. IANA Considerations

This document requests changes to the header and response-code sub-registries of the SIP parameters IANA registry.

8.1 Header Field Names

This document specifies four new SIP headers: Responder, Responder-Info, Response-Authentication and Response-Authorization. Their syntax is given in Section 6. These headers are defined by the following information, which is to be added to the header sub-registry under http://www.iana.org/assignments/sip-parameters.

Header Name: Responder
Compact Form: (none)
Header Name: Responder-Info
Compact Form: (none)
Header Name: Response-Authentication
Compact Form: (none)
Header Name: Response-Authorization
Compact Form: (none)

8.2 431 'Failed Responder Identity Response Code

This document registers a new SIP response code which is described in Section 3.1. It is used when the responder of the SIP response cannot be verified successfully. This response code is defined by the following information, which is to be added to the method and response-code sub-registry under

http://www.iana.org/assignments/sip-parameters.
Response Code Number: 431
Default Reason Phrase: Failed Responder Identity

8.3 432 'Failed Response Authorization Response Code

This document registers a new SIP response code which is described in Section 3.2. It is used when the expected Response-Authorization is missing or doesn't carry the correct digest. This response code is defined by the following information, which is to be added to the method and response-code sub-registry under

http://www.iana.org/assignments/sip-parameters.
Response Code Number: 432
Default Reason Phrase: Bad Identity-Info

9. Acknowledgments

10. Appendix A. AIB used for SIP response identity

The following example is used in this document to demonstrate the mechanisms in many sections:

UAC <------> Proxy-1 <------> Proxy-2 <-----> UAS
UAC: alice@source.com
Proxy-1: px1@source.com
Proxy-2: px2@destination.com
UAS: bob@destination.com

Alice sends an INVITE request to Bob. Proxy-2 receives the request and informs Alice of the response code 183 Session Progress, along with two new header fields called Responder and Responder-Info:

Responder: claimer=px2@destination.com; verify-method=AIB
Responder-Info: https://www.destination.com/certification

For AIB inside S/MIME, some headers including Responder are used as the authenticated body inside S/MIME. It is up to the responder to decide if end-to-end security is needed, which may trigger the encryption of AIB through the public key of the caller, i.e. Alice. AIB is signed with responder's private key to assure its identify.

Assume that TLS is set up for each hop, including between Alice and Proxy-1 and between Proxy-1 and Proxy-2. The mechanism for handling AIB inside S/MIME can be applied for handling the identity in this scenario. Proxy-2 generates the SIP response of 183 Session Progress, and Proxy-2 must insert its URI into Responder with the link to acquire its certification inside Responder-Info.

AIB may be generated by Proxy-2 without any encryption. After verifying AIB for Proxy-2's identify, Proxy-1 can propagate the same info back to Alice. Then Alice can verify Responder by herself through AIB and Responder-Info.

One variation for TLS is that AIB may be encrypted by Proxy-2 with Proxy-1's certificate. This requires Proxy-1 to decrypt the AIB and verify the identity of Proxy-2 . If the identity is proven

consistent, Proxy-1 may have to encrypt the AIB again by Alice's public key. Similarly, Alice can verify the identity of the responder. If the verification fails, Proxy-1 may decide what the right follow-up operations are.

In some scenarios for providing better secure operations, the proxies may verify the identity of the responder. If the verification indicates the unmatched SIP response identity, the proxies may replace the response code with the 431 Failed Responder Identity for announcing the identity problem as early as possible.

If AIB is specified as the verifier-method inside Responder header, AIB inside S/MIME is used to provide the digital signature of the SIP response Identity.

The headers used for this purpose should include the minimum set of To, From, Call-ID, CSeq, Date, and Responder. Any additional headers may be put into AIB by the responder.

The following example is to illustrate the response from Proxy-2. Proxy-2 adds its identity into AIB.

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP px1.source.com;branch=z9hG4bKnashds8
 ;received=127.101.56.17
To: Bob <sip:bob@destination.com>
From: Alice <sip:alice@source.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 50
Date: Thu, 21 Apr 2005 16:28:56 GMT
Responder: claimer=px2@destination.com; verify-method=AIB
Responder-Info: https://www.destination.com/certification
Content-Type: multipart/mixed; boundary=unique-boundary-1

   --unique-boundary-1

   Content-Type: application/sdp
   Content-Length: 147

   v=0
   o=UserA 3569844526 3569844526 IN IP4 source.com
   s=Session SDP
   c=IN IP4 px2.destination.com
   t=0 0
   m=audio 61020 RTP/AVP 0
   a=rtpmap:0 PCMU/8000
```

```
      --unique-boundary-1
      Content-Type: multipart/signed;
        protocol="application/pkcs7-signature";
        micalg=sha1; boundary=boundary68
      Content-Length: 742

      --boundary68
      Content-Type: message/sipfrag
      Content-Disposition: aib; handling=optional

   To: Bob <sip:bob@destination.com>
   From: Alice <sip:alice@source.com>;tag=1928301774
   Call-ID: a84b4c76e66710
   CSeq: 314159 INVITE
   Date: Thu, 21 Apr 2005 16:28:56 GMT
   Responder: claimer=px2@destination.com; verify-method=AIB

      --boundary68
      Content-Type: application/pkcs7-signature; name=smime.p7s
      Content-Transfer-Encoding: base64
                  Content-Disposition: attachment; filename=smime.p7s;
         handling=required

      H77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6vhJhjH776tbB9HG4
      T6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnjTrfvbnj756tbB9HG4VQdT
      hJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4oirDAFqre570
      AFAwqoireikf5287REW

      --boundary42--

      --unique-boundary-1--

      [*digest needs to be recalculated for this message]
```

   It is up to the responder to decide if end-to-end security is needed, which may trigger the encryption of AIB through the public key of the caller. In this case, only the caller can verify the signature of the responder.

11. References

11.1 Normative References

[1] 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.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement

Levels", BCP 14, RFC 2119, March 1997.

[3] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[4] Jennings, C. and J. Peterson, "Certificate Management Service for The Session Initiation Protocol (SIP)", draft-ietf-sipping-certs-01 (work in progress), February 2005.

[5] Peterson, J., "Session Initiation Protocol (SIP) Authenticated Identity Body (AIB) Format", RFC 3893, September 2004.

[6] Metz, C., "OTP Extended Responses", RFC 2243, November 1997.

11.2 Informational References

[7] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[8] Jennings, C., Peterson, J., and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[9] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

Authors' Addresses

Feng Cao
Cisco Systems
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

Email: fcao@cisco.com

Cullen Jennings
Cisco Systems
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 902-3341
Email: fluffy@cisco.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 10, 2006

K. Ono
S. Tachimoto
NTT Corporation

July 9, 2005

End-to-middle Security in the Session Initiation Protocol (SIP)
draft-ietf-sip-e2m-sec-00

Status of this Memo

Copyright Notice

Abstract

Some services provided by intermediaries depend on their ability to inspect a message body in the Session Initiation Protocol (SIP). When sensitive information is included in the message body, a SIP User Agent (UA) needs to protect it from other intermediaries than those that the UA agreed to disclose it to. This document proposes a mechanism for securing information passed between an end user and intermediaries using S/MIME. It also proposes mechanisms for a UA to discover intermediaries which need to inspect an S/MIME-secured

message body, or to receive the message body with data integrity.

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 RFC-2119 [1].

Table of Contents

1. Introduction

When a UA requires services provided by intermediaries that depend on the message body in request/response messages, end-to-end confidentiality currently has to be disabled. This problem is pointed out in Section 23 of [2]. Since such intermediaries are not always adjacent to the UA, this situation requires security between the UA and the intermediaries for the message body. We call this "end-to-middle security", where by "end" we mean a UA and by "middle" we mean an intermediary, typically a proxy server.

End-to-middle security, as well as end-to-end security, consists of peer authentication, data integrity, and data confidentiality. Peer authentication is required to achieve data integrity and data confidentiality respectively. The mechanisms of end-to-middle peer authentication are established with pre-existing mechanisms such as HTTP Digest authentication [7]. Therefore, this document focuses on mechanisms for providing data confidentiality and integrity for end-to-middle security to meet the requirements discussed in [3].

The proposed mechanisms are based on S/MIME [4], since the major requirement is to have little impact on standardized end-to-end security mechanisms, the way of handling S/MIME-secure messages. The mechanisms consist of generating S/MIME-secured message body and indicating the target message body for a proxy server selected by the UA. In addition, this document describes a mechanism for a UA to discover the intermediary which needs to inspect an S/MIME-secured message body, or to receive the message body with data integrity.

2. Generating S/MIME-secured Message Body

2.1 S/MIME-secured Message Body for Confidentiality

For end-to-middle confidentiality, a UA MUST generate S/MIME CMS [5] EnvelopedData. Prior to generating it, a UA needs to identify the target proxy servers and obtain their credentials, such as their public key certificates or shared secrets. One method is shown in Section 4.

The structure of the S/MIME CMS EnvelopedData contains encrypted data specified in the "encryptedContentInfo" field and its recipient list specified in the "recipientInfos" field. The encrypted data is encrypted with a content-encryption-key (CEK) and the recipient list contains the CEKs encrypted with different key-encryption-keys (KEKs), one for each recipient. The KEKs are either the public keys of each recipient or the shared keys between the UA and each recipient.

If the encrypted data is destined for a proxy server, the recipient list MUST contain only the proxy server. If the same encrypted data is destined for multiple proxy servers, or is shared with the user agent server (UAS) and proxy servers, the recipient list MUST be addressed to the proxy servers, or the UAS and the proxy servers. If there are multiple pieces encrypted data destined for each proxy server, the recipient list in each piece of encrypted data MUST contain the relevant proxy server. If a piece of encrypted data is destined for a proxy server and another piece of encrypted data for the UAS, the recipient of each piece of encrypted data MUST be each entity respectively. In order to concatenate more than one CMS EnvelopedData, the user agent client (UAC) MUST generate a multipart MIME body.

For example, a UA uses this mechanism when keying materials, such as keys used for Secure RTP (SRTP), are included in the SDP[8]. Although a proxy server needs to view SDP (i.e., for a firewall traversal service), the UA does not want to show the keying materials to the proxy server. In this case, one CMS EnvelopedData contains the SDP, that includes keying materials of the SRTP stream, encrypted for the UA. The other CMS EnvelopedData contains the SDP, that does not include the keying materials, encrypted for the proxy server.

As described in [2], proxy servers are prohibited from deleting any message body. Even if a UAC send a piece of encrypted data only to a proxy server, the UAS receives it and cannot decrypt it. In order to avoid unnecessary error conditions in the UAS, the UAC MUST set the value "optional" in the handling parameter of the "Content-Disposition" MIME header for the message body. If the multipart MIME body consists of encrypted MIME bodies with the value "optional", the "Content-Disposition" MIME header of the multipart MIME body MUST also contain the value "optional" in the handling parameter. If the multipart MIME body contains a body with the value "required" and another body with the value "optional", the multipart MIME body MUST have either the value "required" in the handling parameter of the "Content-Disposition" MIME header, or no handling parameter, since the default value is "required" as specified in [2]. The UAS SHOULD NOT try to decrypt encrypted data that has the value "optional".

2.2 S/MIME-secured Message Body for Data Integrity

For end-to-middle data integrity, a UA SHOULD generate either S/MIME CMS SignedData. A UA MAY generate a signature in the SIP Identity [9] header, if the UA has its own public and private key. These mechanisms allow any entity to verify the data integrity, if it is able to access the UA's public key. This is why the same mechanisms can be used in both end-to-middle and end-to-end data integrity.

Note: There are other mechanisms which can provide data integrity, such as S/MIME CMS AuthenticatedData, which requires that a UA obtains the credential of the recipient, that is a proxy server, in advance. However, this is not used in [2] and require a mechanism to securely transmit the credential from the proxy server to the UA. Thus, this document does not describe the use of S/MIME CMS AuthenticatedData.

3. Indicating the Target Content

A UA needs a way to indicate content that it expects to be viewed by a proxy server, in order for the proxy server to easily determine whether to process a MIME body and if so, which part. To meet this requirement, the UA SHOULD set a label to indicate the proxy server and its target content using a new SIP header, "Proxy-Required-Body". This header consists of one or more proxy servers' hostnames and one or more "content-id" parameter(s) pointing to the "Content-ID" MIME header placed in the target body.

Note: There were three other options to label a body: a new SIP parameter to an existing SIP header, a new MIME header, or a new parameter to an existing MIME header.
1) Using a new parameter to Route header. Since a proxy server views this header when forwarding a request message, it seems to be a reasonable option. However, it cannot work with strict routing.
2) Using a new MIME header, "Content-Target", as proposed in a previous version of this draft. Since this option is not necessary as a generic mechanism of MIME, it is not preferred.
3) Using a new MIME parameter to "Content-Disposition". The same reason as above.

If a UA needs to label the encrypted data, it SHOULD set the "Proxy-Required-Body" SIP header that contains the address of the proxy server and "content-id" parameter indicating the target S/MIME CMS EnvelopedData.

If a UA needs to label the signed data, it SHOULD set the "Proxy-Required-Body" SIP header that contains the address of the server and "content-id" parameter indicating the S/MIME CMS SignedData. Note that the signature for part of a MIME body alone is meaningless in providing data integrity.

4. Discovering the Security Policies of Proxy Servers

A discovery mechanism for security policies of proxy servers is needed when a UA does not statically know which proxy servers or domains have such policies. Security policies require disclosure of

data and/or verification in order to provide some services which needs UA's compliance.

There are two ways in which a UA can learn the policies of the proxy servers. One is by receiving an error response. A UAC can learn the policies in this way. However, it is not applicable to the UAS because there is no way to react a response message. Alternatively, a policy server can provide a UAC and the UAS a package mentioning proxy's policy as described in [10]. When a proxy server needs to inspect the message body contained in the response, it needs to learn the policies from a policy server before sending the response.

When the proxy server receives a request that can not be accepted due to its condition, the proxy server MUST reject with an error response. If the request contains encrypted data and the proxy server cannot view the message body that has to be viewed in order to proceed, the proxy server MUST reject with a 493 (Undecipherable) error response. The proxy's public key certificate and Content-Type to be viewed SHOULD be contained with the error response. The proxy public key certificate SHOULD be set as an "application/pkix-cert" body. The required Content-Type SHOULD be set in the Warning header with a new warn-code, 380.

If a digital signature is not attached to the message body in the request and the proxy server requires the integrity check, the proxy server MUST reject with a 495 (Signature Required) error response. This error response does not contain signature required Content-Type, since the attached signature to the whole body is always required.

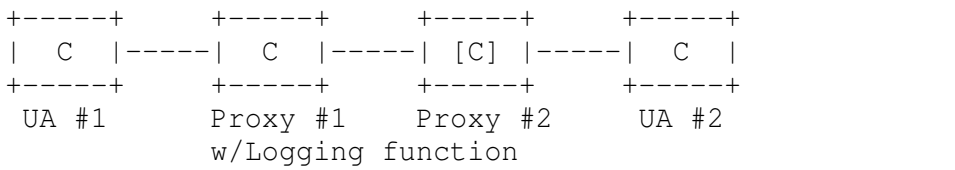
When a proxy server requires both disclosure and an integrity check of the message body in a request message and the message satisfies neither, the proxy server SHOULD send one error response at a time. When a proxy server cannot decrypt the message body in a request message and does not see if the signature is placed inside, a proxy server SHOULD send an error response only for requesting disclosure. After receiving a request message including encrypted data destined for the proxy server, it finds out whether the message has a signature attached and SHOULD send an error response for requesting signature when the message lacks it.

> Note: A 495 (Signature Required) response is not only generated by a proxy server, but also by the UAS.

This discovery mechanism requires two more messages' exchange for an error condition per each proxy server in the signaling path in order to establish a session between UAs. Since this causes a delay in session establishment, it is desirable that the UAs learn the security policies of the proxy servers in advance.

5. Behavior of UAs and Proxy Servers

We describe here an example of the behavior of UAs and proxy servers in a model in which a proxy server that provides a logging service for instant messages exists in a signaling path as shown in Figure 1.



C : Content that UA #1 allows the entities to inspect
[C]: Content that UA #1 prevents the entity from inspecting

Figure 1: Deployment example

5.1 UAC Behavior

When a UAC sends a MESSAGE [11] request including encrypted message content for end-to-end and end-to-middle confidentiality, it MUST use S/MIME CMS EnvelopedData. If UA #1 is unaware of the services provided by Proxy #1 that requires inspecting the message body, UA #1 will MAY get a 493 (Undecipherable) error response and the public key of Proxy #1. After getting the error response, UA #1 MUST use S/MIME CMS EnvelopedData body for UA #2 and Proxy #1. UA #1 SHOULD specify the hostname of Proxy #1 and Content-ID of the S/MIME CMS EnvelopedData to be decrypted by Proxy #1 in the "Proxy-Required-Body" SIP header.

When a UAC sends a request message of which message body needs end-to-middle integrity, it SHOULD use S/MIME CMS SignedData to attach a digital signature. If UA #1 does not know the service of Proxy #1 that requires verifying the message body, UA #1 MAY get a 495 (Signature Required) error response. After getting the error response, UA #1 SHOULD generate the CMS SignedData to attach a signature by computing with its own private key. UA #1 SHOULD specify the hostname of Proxy #1 and Content-ID of the CMS SignedData to be validated by Proxy #1 in the "Proxy-Required-Body" SIP header.

When a UAC sends a request and needs both end-to-middle confidentiality and integrity for the message body, it SHOULD first attach a digital signature, and then encrypted the message body. In this example, UA #1 SHOULD generate S/MIME CMS SignedData for the contents, and then generate S/MIME CMS EnvelovedData body to encrypt the CMS SignedData. UA#1 SHOULD specify the hostname of Proxy#1 and Content-IDs of the CMS SignedData and the CMS EnvelopedData destined

for Proxy #1 in the "Proxy-Required-Body".

When a UAC generates S/MIME CMS EnvelopedData, the UAC MAY use the CEK reuse mechanism [12][13]. The CEK reuse mechanism has a benefit that enables UAs to efficiently encrypt/decrypt data in subsequent messages. The UAC MAY use the "unprotectedAttrs" field to stipulate reuse of the CEK and indicate its identifier. When the UAC reuses the CEK in the previous request as the KEK, it generates CMS EnvelopedData with the type "KEKRecipientInfo" of "RecipientInfo" attribute.

5.2 UAS Behavior

When the UAS receives a request that uses S/MIME, it first decrypts and/or validates the S/MIME bodies as usual. In particular, when the CMS EnvelopedData body of the request contains the "unprotectedAttrs" attribute specifying reuse of the CEK, the UAS MAY keep the CEK with the identifier specified in the "unprotectedAttrs" attribute.

When the UAS responds with a 200 OK, the same type of S/MIME CMS data is RECOMMENDED to be used. For example, if the UAS receives an INVITE request in which the SDP is encrypted by using the CMS EnvelopedData, it is RECOMMENDED to respond with a 200 OK response in which the SDP is encrypted by using the CMS EnvelopedData body. If the UAS receives an INVITE request which is attached a digital signature to the SDP by using the CMS SignedData, it is RECOMMENDED to respond with a 200 OK response which is attached a signature to the SDP by using the CMS SignedData. In the above example, however, a 200 OK response to the MESSAGE request does not need to use the same type of S/MIME CMS data since the response does not contain any MIME body.

Even when the UAS receives a request that does not use S/MIME, the UAS sometimes needs end-to-end and end-to-middle confidentiality for the message body and/or headers in a response. In this case, the UAS MUST use CMS EnvelopedData to encrypt it. When the UAS sends a response and needs end-to-end and end-to-middle integrity for the message body and/or headers, it SHOULD use CMS SignedData to attach a digital signature. This is not different from how a UAC operates as described in Section 5.1.

5.3 Proxy Behavior

When a proxy server supporting this mechanism receives a message, it MUST inspect the "Proxy-Required-Body" header. If the header includes the processing server's own hostname, the proxy server MUST inspect the body specified by the Content-ID. When the specified body is CMS EnvelopedData, the proxy server MUST inspect it and try

to decrypt the "recipientInfos" field. If the header does not include the server's own name, nor the header exists, the proxy server MAY view the message body.

If there is a piece of encrypted data for the proxy, the proxy server will succeed in decryption using the "recipientInfos" field. If the proxy server fails to decrypt the message body that is required to view, it MUST respond with a 493 (Undecipherable) response if it is a request, otherwise any existing dialog MUST be terminated.

If the proxy server succeeds in this decryption, it MAY inspect the "unprotectedAttrs" field of the CMS EnvelopedData body. If the attribute gives the key's identifier, the proxy server MAY keep the CEK with its identifier until the lifetime of the CEK expires. If it receives subsequent messages within the lifetime, it MAY try to decrypt the type "KEKRecipientInfo" of the "RecipientInfo" attribute by using this CEK.

When the specified content is CMS SignedData body, the proxy server MUST inspect it and validate the digital signature. If the verification fails, the proxy server SHOULD reject the subsequent procedure and respond with a 495 (Signature Required) response if the message is a request, otherwise any existing dialog MAY be terminated.

When the proxy server forwards the request, it modifies the routing headers as it normally does, but does not modify the message body. The proxy server MAY delete the "Proxy-Required-Body" header that contains its own hostname.

When a provider operating the proxy server does not allow any information related to its security policies to be revealed to the proxy server serving the recipient UA, the proxy server deletes the "Proxy-Required-Body" header. However, when a request message is transmitted to the proxy server via a proxy server operated by another provider, there is no way to conceal the header from the other proxy servers.

If a proxy does not support this mechanism and receives a message with the "Proxy-Required-Body" header, the proxy MUST ignore the header and operate as usual.

6. Proxy-Required-Body Header Field Use

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [6]. The new header "Proxy-Required-Body" is defined as a SIP header.

```
Proxy-Required-Body   = "Proxy-Required-Body" HCOLON required-proxy
                        SEMI target-body
required-proxy        = host
target-body           = cid-param *(COMMA cid-param)
cid-param             = "cid" EQUAL content-id
content-id            = LDQUOT dot-atom "@" (dot-atom / host) RDQUOT
dot-atom              = atom *( "." atom )
atom                  = 1*( alphanum / "-" / "!" / "%" / "*" /
                        "_" / "+" / "'" / "`" / "~"   )
```

Information about the use of headers in relation to SIP methods and proxy processing is summarized in Table 1.

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Proxy-Required-Body	R	dr	-	o	-	o	o	o
Proxy-Required-Body	100-699	dr	-	o	-	o	o	o

Header field	where	proxy	SUB	NOT	PRK	IFO	UPD	MSG
Proxy-Required-Body	R	dr	o	o	-	o	o	o
Proxy-Required-Body	100-699	dr	o	o	-	o	o	o

Table 1: Summary of header field use

The "where" column gives the request and response types in which the header field can be used. The values in the "where" column are as follows:

* R: The header field may appear in requests
* 100-699: A numeral range indicates response codes with which the header field can be used.

The "proxy" column gives the operations a proxy may perform on the header field:

* d: A proxy can delete a header field value.
* r: A proxy must be able to read the header field, so it cannot be encrypted.

The next columns relate to the presence of a header field in a method:

* o: The header field is optional.
* -: The header field is not applicable.

7. Message Examples

The following examples illustrate the use of the mechanism defined in the previous sections.

7.1 Message Examples of End-to-Middle Confidentiality

In the following example, a UAC needs message content in a MESSAGE

request to be confidential and it allows a proxy server to view the message body. It also needs to reuse the CEK in the subsequent request messages. Even though the Content-Length has no digit, the appropriate length is to be set. In the example message below, the text with the box of asterisks ("*") is encrypted:

```
MESSAGE alice@atlanta.example.com --> ss1.atlanta.example.com


MESSAGE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
Route: <sip:ss1.atlanta.example.com;lr>
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 MESSAGE
Date: Fri, 20 June 2003 13:02:03 GMT
Proxy-Required-Body: ss1.atlanta.example.com;
                     cid=1234@atlanta.example.com

Content-Type: application/pkcs7-mime;smime-type=enveloped-data;
              name=smime.p7m
Content-Transfer-Encoding: binary
Content-ID: 1234@atlanta.example.com
Content-Disposition: attachment;filename=smime.p7m;handling=required
Content-Length: ...

**********************************************************************
* (encryptedContentInfo)                                             *
* Content-Type: text/plain                                           *
* Content-Length: ...                                                *
*                                                                    *
* Hello.                                                             *
* This is confidential.                                              *
*                                                                    *
* (recipientInfos)                                                   *
* RecipientInfo[0] for ss1.atlanta.example.com public key            *
* RecipientInfo[1] for Bob's public key                              *
*                                                                    *
* (unprotectedAttrs)                                                 *
* CEKReference                                                       *
**********************************************************************
```

If the proxy server successfully views the message body, the UAC receives a 200 OK from the UAS normally. However, if a proxy server fails to view the message body, the UAC receives a 493

(Undecipherable) error response from the proxy server, as follows:

493 Undecipherable alice@atlanta.example.com <--
ss1.atlanta.example.com

```
SIP/2.0 493 Undeciperable
Warning: 380 ss1.atlanta.example.com "Required to view 'text/plain'"
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
;received=192.0.2.101
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 MESSAGE
Content-Type: application/pkix-cert
Content-Length: ...

<certificate>
```

In the following example, a UA needs the SDP in an INVITE request to be confidential and it allows a proxy server to view the SDP. It also needs to reuse the CEK of the encrypted data in the subsequent request messages.

```
INVITE alice@atlanta.example.com --> ss1.atlanta.example.com


INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Date: Fri, 20 June 2003 13:02:03 GMT
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Proxy-Required-Body: ss1.atlanta.example.com;
                     cid=1234@atlanta.example.com

Content-Type: application/pkcs7-mime;smime-type=enveloped-data;
              name=smime.p7m
Content-Transfer-Encoding: binary
Content-ID: 1234@atlanta.example.com
Content-Disposition: attachment;filename=smime.p7m;handling=required
Content-Length: ...

*****************************************************************
* (encryptedContentInfo)                                        *
* Content-Type: application/sdp                                 *
* Content-Length: 151                                           *
*                                                               *
* v=0                                                           *
* o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com*
* s=-                                                           *
* c=IN IP4 192.0.2.101                                          *
* t=0 0                                                         *
* m=audio 49172 RTP/AVP 0                                       *
* a=rtpmap:0 PCMU/8000                                          *
*                                                               *
* (recipientInfos)                                              *
* RecipientInfo[0] for ss1.atlanta.example.com public key       *
* RecipientInfo[1] for Bob's public key                         *
*                                                               *
* (unprotectedAttrs)                                            *
* CEKReference                                                  *
*****************************************************************
```

When the proxy server successfully views the SDP, and the UAS responds with a 200 OK. The 200 OK is to be encrypted as follows:

```
   200 OK alice@atlanta.example.com <-- ss1.atlanta.example.com


   SIP/2.0 200 OK
   Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
       ;received=192.0.2.101
   From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 1 INVITE
   Contact: <sip:bob@client.biloxi.example.com;transport=tcp>
   Content-Type: application/pkcs7-mime;smime-type=enveloped-data;
                 name=smime.p7m
   Content-Transfer-Encoding: binary
   Content-ID: 1234@atlanta.example.com

   ************************************************************
   * (encryptedContentInfo)                                   *
   * Content-Type: application/sdp                            *
   * Content-Length: 147                                      *
   *                                                          *
   * v=0                                                      *
   * o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com*
   * s=-                                                      *
   * c=IN IP4 192.0.2.201                                     *
   * t=0 0                                                    *
   * m=audio 3456 RTP/AVP 0                                   *
   * a=rtpmap:0 PCMU/8000                                     *
   *                                                          *
   * (recipientInfos)                                         *
   * RecipientInfo[0] for Alice's public key                  *
   ************************************************************
```

7.2 Message Examples of End-to-Middle Integrity

In the following example, a UA needs the integrity of message content in a MESSAGE request to be validated by a proxy server before it views message content. Even though the Content-Length has no digit, the appropriate length is to be set.

```
MESSAGE alice@atlanta.example.com --> ss1.atlanta.example.com


MESSAGE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
Route: <sip:ss1.atlanta.example.com;lr>
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 MESSAGE
Date: Fri, 20 June 2003 13:02:03 GMT
Proxy-Required-Body: ss1.atlanta.example.com;
                     cid=1234@atlanta.example.com
Content-Type: multipart/signed;protocol="application/pkcs7-signature"
         ;micalg=sha1;boundary=boundary1
Content-Length: ...

--boundary1
Content-Type: text/plain
Content-Length: ...

Hello.
This is protected with the signature.
--boundary1
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: binary
Content-ID:1234@atlanta.example.com
Content-Disposition: attachment;
filename=smime.p7s;handling=required

[binary data]
--boundary1--
```

If the proxy server successfully validates the integrity of the message body, the UAC normally receives a 200 OK from the UAS. However, if a proxy server does not receive a signature for the whole message body, the UAC receives a 495 (Signature Required) error response from the proxy server, as follows:

```
   495 Signature Required alice@atlanta.example.com <--
   ss1.atlanta.example.com


   SIP/2.0 495 Signature Required
   Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
   ;received=192.0.2.101
   From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
   To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
   Call-ID: 3848276298220188511@atlanta.example.com
   CSeq: 1 MESSAGE
   Content-Length: 0
```

8. Security Considerations

8.1 Impersonating a Proxy Server

In the discovery mechanism in Section 4, a UA receives a 493 (Undecipherable) error response with the public key certificate of the proxy server requesting the disclosure of the message body. The public key certificate in the error response is vulnerable to be forged by a malicious user.

To make sure that the response is sent by a proper proxy server, a UA needs to authenticate the response. Since the UA is not always adjacent to the proxy server, the UA cannot directly authenticate the proxy server by security mechanisms of the transport layer or the below. A UA SHOULD verify the chains to a trusted certificate authority of the public key certificate.

8.2 Tampering with a Message Body

This document describes a mechanism to encrypt data for multiple recipients, such as multiple proxy servers, or a recipient UA and proxy servers. A piece of encrypted data is decipherable and vulnerable to tampering by proxy servers at the previous hops.

In order to prevent such tampering, the UA SHOULD protect the data integrity before encryption, when the encrypted data is meant to be shared with multiple proxy servers, or to be shared with the UAS and selected proxy servers. The UA SHOULD generate S/MIME CMS SignedData and then SHOULD generate the EnvelopedData to encrypt attached data with a digital signature. The recipient entity SHOULD verify the signature to see if the encrypted data has been modified after decryption by an entity listed in the "recipientInfos" field.

8.3 Tampering with the Label of the Target Content

This document also describes a new SIP header for labeling a message body for a proxy server. If a malicious user or proxy server modified/added/deleted the label, the specified message body is not inspected by the specified proxy server, and some services requiring its content can not be provided. Or a proxy server will conduct an unnecessary processing on message bodies such as unpacking MIME structure, and/or signature verification. This is a possible cause for a Denial-of-Services attack to a proxy server.

To prevent such attacks, data integrity for the label is needed. UAs and proxy servers SHOULD use TLS mechanism to communicate with each other. Since a proxy server trusted to provide SIP routing is basically trusted to process SIP headers other than those related to routing, hop-by-hop security is reasonable to protect the label. In order to further protect the integrity of the label, UAs MAY generate a "message/sipfrag" body and attach a digital signature for the whole body.

9. IANA Considerations

This document defines a new SIP header, "Proxy-Required-Body", of which the syntax is shown in Section 6. This document also defines a new SIP response-code, 495 "Signature Required", and a new Warn-code, 380 "Required to view Content-Type", as described in Section 4.

10. Acknowledgments

Thanks to Rohan Mahy and Cullen Jennings for their initial support of this concept and to many people for useful comments, especially Jon Peterson, Jonathan Rosenberg, Eric Burger, and Russ Housely.

11. References

11.1 Normative References

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

[2] 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.

[3] Ono, K. and S. Tachimoto, "Requirements for end-to-middle security in the Session Initiation Protocol (SIP)", draft-ietf-sipping-e2m-sec-reqs-06 (work in progress), March 2005.

[4] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling", RFC 3850, July 2004.

[5] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[6] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

11.2 Informative References

[7] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[8] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol Security Descriptions for Media Streams", draft-ietf-mmusic-sdescriptions-11 (work in progress), June 2005.

[9] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[10] Hilt, V., Camarillo, G., and J. Rosenberg, "Session Initiation Protocol (SIP) Session Policies - Document Format and Session-Independent Delivery Mechanism", draft-ietf-sipping-session-indep-policy-02 (work in progress), February 2005.

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

[12] Farrell, S. and S. Turner, "Reuse of CMS Content Encryption Keys", RFC 3185, October 2001.

[13] Ono, K. and S. Tachimoto, "Key reuse in S/MIME for SIP", draft-ono-sipping-keyreuse-smime-00 (work in progress), February 2004.

[14] Sparks, R., "Internet Media Type message/sipfrag", RFC 3420, November 2002.

Authors' Addresses

Kumiko Ono
Network Service Systems Laboratories, NTT Corporation
Musashino-shi, Tokyo 180-8585
Japan

Email: ono.kumiko@lab.ntt.co.jp

Shinya Tachimoto
Network Service Systems Laboratories, NTT Corporation
Musashino-shi, Tokyo 180-8585
Japan

Email: tachimoto.shinya@lab.ntt.co.jp

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 15, 2006

J. Rosenberg
Cisco Systems

July 14, 2005

Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)
draft-ietf-sip-gruu-04

Status of this Memo

Copyright Notice

Abstract

Several applications of the Session Initiation Protocol (SIP) require a user agent (UA) to construct and distribute a URI which can be used by anyone on the Internet to route a call to that specific UA instance. A URI which routes to a specific UA instance is called a Globally Routable UA URI (GRUU). This document describes an extension to SIP for obtaining a GRUU from a server, and for communicating a GRUU to a peer within a dialog.

Table of Contents

1. Introduction

The Session Initiation Protocol, RFC 3261 [1] is used to establish and maintain a dialog between a pair of user agents in order to manage a communications session. Messages within the dialog are sent from one user agent to another using a series of proxy hops called the route set, eventually being delivered to the remote target - the user agent on the other side of the dialog. This remote target is a SIP URI obtained from the value of the Contact header field in INVITE requests and responses.

RFC 3261 mandates that a user agent populate the Contact header field in INVITE requests and responses with a URI that is global (meaning that it can be used from any element connected to the Internet), and that routes to the user agent which inserted it. RFC 3261 also mandates that this URI be valid for requests sent outside of the dialog in which the Contact URI was inserted.

In practice, these requirements have proven very difficult to meet. Endpoints often have only an IP address and not a hostname that is present in DNS, and this IP address is frequently a private address, because the client is behind a NAT. Techniques like the Simple Traversal of UDP Through NAT (STUN) [15] can be used to obtain IP addresses on the public Internet. However, many firewalls will prohibit incoming SIP requests from reaching a client unless they first pass through a proxy sitting in the DMZ of the network. Thus URIs using STUN-obtained IP addresses often do not work.

Because of these difficulties, most clients have actually been inserting URIs into the Contact header field of requests and responses with the form sip:<IP-address>. These have the property of routing to the client, but they are generally only reachable from the proxy to which the user is directly connected. This limitation does not prevent normal SIP calls from proceeding, since the user's proxy can usually reach these private addresses, and the proxy itself is generally reachable over the public network. However, this issue has impacted the ability of several other SIP mechanisms and applications to work properly.

An example of such an application is call transfer [24], based on the REFER method [7]. Another application is the usage of endpoint-hosted conferences within the conferencing framework [17]. Both of these mechanisms require the endpoint to be able to construct a URI that not only routes to that user agent, but is usable by other entities anywhere on the Internet as a target for new SIP requests.

This specification formally defines a type of URI called a Globally Routable User Agent URI (GRUU) which has the properties of routing to

the UA and being reachable from anywhere. Furthermore, it defines a new mechanism by which a client can obtain a GRUU from its SIP provider, allowing it to use that URI in the Contact header fields of its dialog forming requests and responses. Since the GRUU is provided by the user's SIP provider, the GRUU properties can be guaranteed by the provider. As a result, the various applications which require the GRUU property, including transfer, presence, and conferencing, can work reliably.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [5] and indicate requirement levels for compliant implementations.

This specification also defines the following additional terms:

contact: The term "contact", when used in all lowercase, refers to a URI that is bound to an AOR or GRUU by means of a registration. A contact is usually a SIP URI, and is bound to the AOR and GRUU through a REGISTER request by appearing as the value of the Contact header field.

remote target: The term "remote target" refers to a URI that a user agent uses to identify itself for receipt of subsequent requests mid-dialog. A remote target is established by placing a URI in the Contact header field of a dialog forming request or response.

Contact header field: The term "Contact header field", with a capitalized C, refers to the header field which can appear in REGISTER requests and responses, redirects, or in dialog creating requests and responses. Depending on the semantics, the Contact header field sometimes conveys a contact, and sometimes conveys a remote target.

3. Defining a GRUU

URIs have properties. Those properties are granted to the URI based on the policies of the domain that owns the URI, and those properties are not visible by inspection of the URI. Some of the properties that a domain can confer upon a URI are:

The AOR property: A URI has the Address of Record (AOR) property if a domain will allow it to appear in the To header field of REGISTER request.

The alias property: A URI is an alias if its treatment by the domain is identical to another URI.

The service treatment property: A URI has the service treatment property if the domain will apply applications, features, and services to calls made by, or made to, that URI, possibly based on associating that URI with a user that has "subscribed" to various features.

The anonymous property: A URI has the anonymous property when it is not possible, by inspection of the URI, to discern the user with whom the URI is associated.

The identity property: A URI is considered an identity when it is one that the domain will authorize as a valid value in the From header field of a request, such that an authentication service will sign a request with that URI [19].

This specification focuses on a property, called the Globally Routable User Agent URI (GRUU) property. A URI possesses this property when the following is true:

Global: It can be used by any UAC connected to the Internet. In that regard, it is like the address-of-record (AOR) property. A URI with the AOR property (for example, sip:joe@example.com), is meant to be used by anyone to reach that user. The same is true for a URI with the GRUU property.

Routes to a Single Instance: A request sent to that URI will be routed to a specific UA instance. In that regard, it is unlike the address-of-record property. When a request is sent to a URI with the AOR property, routing logic is applied in proxies to deliver the request to one or more UAs. That logic can result in a different routing decision based on the time-of-day, or the identity of the caller. However, when a request is made to a URI with the GRUU property, the routing logic is dictated by the GRUU property. The request has to be delivered to a very specific UA instance. That UA instance has to be the same UA instance for all requests sent to that URI.

Long Lived: The URI with the GRUU property persists for relatively long periods of time, ideally being valid for the duration of existence of the AOR itself. This property cannot be completely guaranteed, but providers are supposed to do their best to make sure that a GRUU remains viable indefinitely.

A URI can have any combination of these properties. It is the responsibility of the domain which mints the URI to determine what

properties are conferred upon that URI. This specification imposes requirements on a domain that mints a URI with the GRUU property.

For convenience, a URI that possesses the GRUU property is also referred to as a GRUU.

4. Use Cases

There are several use cases where the GRUU properties are truly needed in order for a SIP application to operate.

4.1 REFER

Consider a blind transfer application [24]. User A is talking to user B. User A wants to transfer the call to user C. So, user A sends a REFER to user C. That REFER looks like, in part:

```
REFER sip:C@example.com SIP/2.0
From: sip:A@example.com;tag=99asd
To: sip:C@example.com
Refer-To: (URI that identifiers B's UA)
```

The Refer-To header field needs to contain a URI that can be used by user C to place a call to user B. However, this call needs to route to the specific UA instance which user B is using to talk to user A. If it didn't, the transfer service would not execute properly. This URI is provided to user A by user B. Because user B doesn't know who user A will transfer the call to, the URI has to be usable by anyone. Therefore, it needs to be a GRUU.

4.2 Conferencing

A similar need arises in conferencing [17]. In that framework, a conference is described by a URI which identifies the focus of the conference. The focus is a SIP UA that acts as the signaling hub for the conference. Each conference participant has a dialog with the focus. One case described in the framework is where a user A has made a call to user B. User A puts user B on hold, and calls user C. Now, user A has two separate dialogs for two separate calls - one to user B, and one to user C. User A would like to conference them. To do this, user A's user agent morphs itself into a focus. It sends a re-INVITE or UPDATE [4] on both dialogs, and provides user B and user C with an updated remote target that now holds the conference URI. The URI in the Contact header field also has a callee capabilities [11] parameter which indicates that this URI is a conference URI. User A proceeds to mix the media streams received from user B and user C. This is called an ad-hoc conference.

At this point, normal conferencing features can be applied. That means that user B can send another user, user D, the conference URI, perhaps in an email. User D can send an INVITE to that URI, and join the conference. For this to work, the conference URI used by user A in its re-INVITE or UPDATE has to be usable by anyone, and it has to route to the specific UA instance of user A that is acting as the focus. If it didn't, basic conferencing features would fail. Therefore, this URI has to be a GRUU.

4.3 Presence

In a SIP-based presence [25] system, the Presence Agent (PA) generates notifications about the state of a user. This state is represented with the Presence Information Document Format (PIDF) [23]. In a PIDF document, a user is represented by a series of tuples, each of which describes the services that the user has. Each tuple also has a URI in the <contact> element, which is a SIP URI representing that device. A watcher can make a call to that URI, with the expectation that the call is routed to the service whose presence is represented in the tuple.

In some cases, the service represented by a tuple may exist on only a single user agent associated with a user. In such a case, the URI in the presence document has to route to that specific UA instance. Furthermore, since the presence document could be used by anyone who subscribes to the user, the URI has to be usable by anyone. As a result, it has to be a GRUU.

It is interesting to note that the GRUU may need to be constructed by a presence agent, depending on how the presence document is computed by the server.

5. Overview of Operation

This section is tutorial in nature, and does not specify any normative behavior.

This extension allows a UA to obtain a GRUU, and to use a GRUU. These two mechanisms are separate, in that a UA can obtain a GRUU in any way it likes, and use the mechanisms in this specification to use them. This specification defines two mechanisms for obtaining a GRUU - through registrations, and through administrative operation. Only the former requires protocol operations.

A UA can obtain a GRUU by generating a normal REGISTER request, as specified in RFC 3261 [1]. This request contains a Supported header field with the value "gruu", indicating to the registrar that the UA supports this extension. The UA includes a "sip.instance" media

feature tag in the Contact header field of each contact for which a GRUU is desired. This media feature tag contains a globally unique ID that identifies the UA instance. If the domain that the user is registering against also supports GRUU, the REGISTER responses will contain the "gruu" parameter in each Contact header field. This parameter contains a GRUU which the domain guarantees will route to that UA instace. The GRUU is associated with the UA instace. Should the client change its contact, but indicate that it represents the same instance ID, the server would provide the same GRUU. Furthermore, if the registration for the contact expires, and the UA registers the contact at a later time with the same instance identifier, the server would provide the same GRUU.

Since the GRUU is a URI like any other, it can be handed out by a UA by placing it in any header field which can contain a URI. A UA will place the GRUU into the Contact header field of dialog creating requests and responses it generates; RFC 3261 mandates that the Contact header field have the GRUU property, and this specification provides a reliable way for a UA to obtain one. In other words, clients use the GRUU as a remote target. However, since the remote target used by clients to date has typically not had the GRUU properties, implementations have adapted their behaviors (oftentimes in proprietary ways) to compensate. To facilitate a transition away from these behaviors, it is necessary for a UA receiving the message to know whether the remote target is a GRUU or not. To make this determination, the UA looks for the presence of the Supported header field in the request or response. If it is present with a value of "gruu", it means that the remote target is a GRUU.

A domain can construct a GRUU in any way it chooses. However, it is sometimes desirable to construct them in a way which allows for any entity that receives the GRUU to determine the AOR for the subscriber associated with the UA instance. To facilitate that, the GRUU can be constructed by adding the "opaque" URI parameter to the subscriber's AOR. This parameter would contain the context needed for the domain to recognize and treat the URI as a GRUU.

When a UA uses a GRUU, it has the option of adding the "grid" URI parameter to the GRUU. This parameter is opaque to the proxy server handling the domain. However, when the server maps the GRUU to the contact bound to it, the server will add the grid parameter into the registered contact, and use the result in the Request URI. As a result, when the UA receives the request, the Request URI will contain the grid parameter it placed in the corresponding GRUU.

The "grid" and "opaque" URI parameters play similar roles, but complement each other. The "opaque" parameter is added by the owner of the domain in order to ensure that the URI has the GRUU property.

The "grid" parameter is added by the UA instance so that, when a request is received by that instance, it can determine the context of the request.

6. Creation of a GRUU

A GRUU is a URI that is created and maintained by a server authoritative for the domain in which the GRUU resides. Independently of whether the GRUU is created as a result of a registration or some other means, a server maintains certain information associated with the GRUU. This information, and its relationship with the GRUU, is modeled in Figure 2.

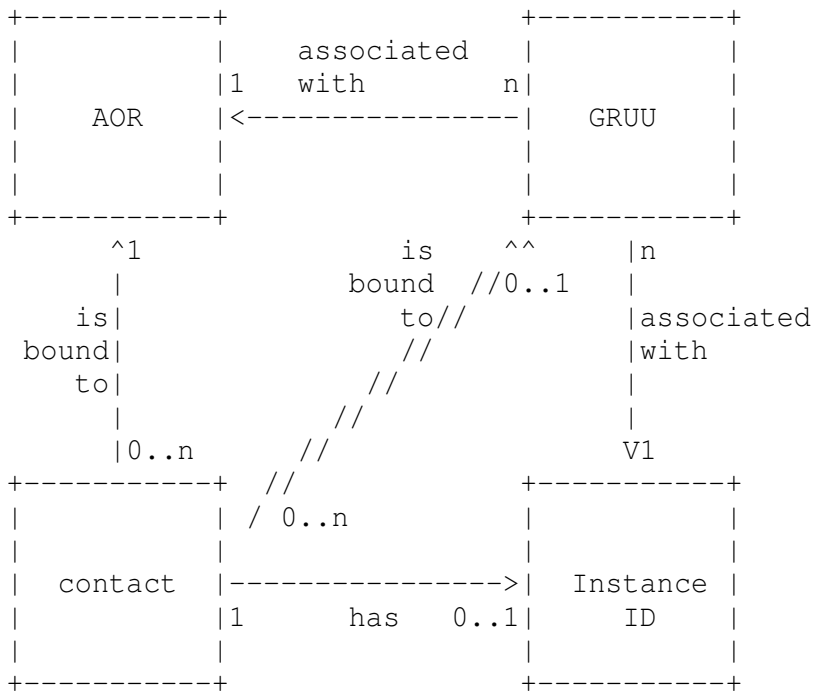


Figure 2

The instance ID plays a key role in this specification. It is an identifier, represented as a URN, that uniquely identifies a SIP user agent amongst all other user agents associated with an AOR. For hardware-based user agents, the instance ID would typically be burned into the device in the factory, similar to the way a unique serial

number is encoded into each device. For software-based user agents, each installation represents a unique instance. As such, the identifier could be generated on installation and then stored on disk for persistence.

A GRUU is associated, in a one-to-one fashion, with the combination of an AOR and instance ID. This combination is referred to as an instance ID/AOR pair. For each GRUU, there is one instance ID/AOR pair, and for each instance ID/AOR pair, there is one GRUU. The instance ID/AOR pair serves to uniquely identify a user agent instance servicing a specific AOR. The AOR identifies a resource, such as a user or service within a domain, and the instance ID identifies a specific UA instance servicing requests for that resource.

It is important to understand that GRUU is associated with the instance ID/AOR pair, not just the instance ID. For example, if a user registered the contact sip:ua@pc.example.com to the AOR sip:user@example.com, and included a +sip.instance="urn:foo:1" parameter in the Contact header field, and also registered the contact sip:ua-112@pc.example.com with the same +sip.instance Contact header field parameter to a second AOR, say sip:boss@example.com, each of those UA instances would have a different GRUU, since they belong to different AORs. That is the reason why a single instance ID can be associated with multiple GRUU; there would be one such association for each AOR. The same goes for the association of AOR to GRUU; there would be one such association for each instance ID.

In many ways, a GRUU is a parallel to an AOR. A URI cannot have both the AOR property and the GRUU property. Just as a contact can be bound to an AOR, a contact can be bound to a GRUU. Any number of contacts can be bound to an AOR, but only those contacts for a particular instance are bound to the GRUU. As discussed in Section 8.4.1 If there are more than one contacts of a particular instance bound to the AOR, only the most recently registered one is used. Similarly, if there are more than one contacts of a particular instance bound to the GRUU, only the most recently registered one is used. Using only the most recently registered contact from an instance ensures that, upon failure and reboot, an instance that obtains and registers a new IP address immediately renders its previous one inactive. Multiple active registrations from a single instance is useful for certain high availability scenarios, and mechanisms for achieving that using a GRUU are described in [18].

The contacts that are bound to the GRUU are always the ones that have an instance ID associated with that GRUU. If none of the contacts bound to the AOR have the instance ID associated with the GRUU, then there are no contacts bound to the GRUU. If a contact should become

registered to the AOR that has an instance ID equal to the one associated with the GRUU, that contact also becomes bound to the GRUU. If that contact should expire, it will no longer be bound to the AOR, and similarly, it will no longer be bound to the GRUU. The URI of the contact is irrelevant in determining whether it is bound to a particular GRUU; only the instance ID and AOR are important.

This specification does not mandate a particular mechanism for construction of the GRUU. Several example approaches are given in Appendix A. However, the GRUU MUST exhibit the following properties:

- The domain part of the URI is an IP address present on the public Internet, or, if it is a host name, the resolution procedures of RFC 3263 [2], once applied, result in an IP address on the public Internet.

- When a request is sent to the GRUU, it routes to a server that can make sure the request is delivered to the UA instance. For GRUU created through registrations, this means that the GRUU has to route to a proxy server with access to registration data.

- A server in the domain can determine that the URI is a GRUU.

- For each GRUU, both the SIP and SIPS versions MUST exist.

Section 8.4 defines additional behaviors that a proxy must exhibit on receipt of a GRUU.

When a domain constructs a URI with the GRUU properties, it MAY confer other properties upon this URI as a matter of domain policy. Of course, the AOR property cannot also be provided, since the GRUU and AOR properties are mututally exclusive. However, a domain can elect to confer properties like identity, anonymity, and service treatment. There is nothing in this specification that can allow the recipient of the GRUU to determine which of these properties besides the GRUU property itself have been conferred to the URI.

The service treatment property merits further discussion. Typically, the services a proxy executes upon receipt of a request sent to a GRUU will be a subset of those executed when a request is sent to the AOR. For requests that are outside of a dialog, it is RECOMMENDED to apply screening types of functions, both automated (such as black and white list screening) and interactive (such as interactive voice response (IVR) applications which confer with the user to determine whether to accept a call). However, forwarding services, such as call forwarding, SHOULD NOT be provided for requests sent to a GRUU. The intent of the GRUU is to target a specific UA instance, and this is incompatible with forwarding operations.

Mid-dialog requests will also be sent to GRUUs, as they are included as the remote-target in dialog forming requests and responses. In those cases, however, a proxy SHOULD only apply services that are meaningful for mid-dialog requests generally speaking. This excludes screening functions, as well as forwarding ones.

The "opaque" URI parameter, defined in Section 9 provides a means for a domain to construct a GRUU such that the AOR associated with the GRUU is readily extractable from the GRUU. Unless the GRUU is meant to also possess the anonymity property, it is RECOMMENDED that GRUUs be constructed using this parameter.

Since the GRUU is associated with both the instance ID and AOR, for any particular AOR there can be a potentially infinite number of GRUU, one for each instance ID. However, the instance IDs are only known to the network when an instance actually registers with one. As a result, it is RECOMMENDED that a GRUU exist from the time a contact with an instance ID is first registered to an AOR, until the time that the AOR is no longer valid in the domain. In this context, the GRUU exists if the domain, upon receiving a request for that GRUU, recognizes it as a GRUU, can determine the AOR and instance ID associated with it, and translate the GRUU to a contact if there is one with that instance ID currently registered. This property of the GRUU can be difficult to achieve through software failures and power outages within a network, and for this reason, the requirement is at RECOMMENDED strength, and not MUST.

7. Obtaining a GRUU

A GRUU can be obtained in many ways. This document defines two - through registrations, and through administrative operation.

7.1 Through Registrations

When a GRUU is associated with a user agent that comes and goes, and therefore registers to the network to bind itself to an AOR, a GRUU is provided to the user agent through SIP REGISTER messages.

7.1.1 User Agent Behavior

7.1.1.1 Generating a REGISTER Request

When a UA compliant to this specification generates a REGISTER request (initial or refresh), it MUST include the Supported header field in the request. The value of that header field MUST include "gruu" as one of the option tags. This alerts the registrar for the domain that the UA supports the GRUU mechanism.

Furthermore, for each contact for which the UA desires to obtain a GRUU, the UA MUST include a "sip.instance" media feature tag as a UA characteristic [11]. As described in [11], this media feature tag will be encoded in the Contact header field as the "+sip.instance" Contact header field parameter. The value of this parameter MUST be a URN [10]. [11] defines equality rules for callee capabilities parameters, and according to that specification, the "sip.instance" media feature tag will be compared by case sensitive string comparison. This means that the URN will be encapsulated by angle brackets ("<" and ">") when it is placed within the quoted string value of the +sip.instance contact parameter. The case sensitive matching rules apply only to the generic usages defined there and in the caller preferences specification [22]. When the instance ID is used in this specification, it is effectively "extracted" from the value in the "sip.instance" media feature tag, and thus equality comparisons are performed using the rules for URN equality specific to the scheme in the URN. If the element performing the comparisons does not understand the URN scheme, it performs the comparisons using the lexical equality rules defined in RFC 2141. Lexical equality may result in two URN being considered unequal when they are actually equal. In this specific usage of URNs, the only element which provides the URN is the SIP UA instance identified by that URN. As a result, the UA instance SHOULD provide lexically equivalent URNs in each registration it generates. This is likely to be normal behavior in any case; clients are not likely to modify the value of the instance ID so that it remains functionally equivalent to previous registrations, but lexigraphically different.

This specification makes no normative recommendation on the specific URN that is to be used in the "+sip.instance" Contact header field parameter. However, the URI MUST be selected such that the instance can be certain that no other instance registering against the same AOR would choose the same URI value. Usage of a URN is a MUST since it provides a persistent and unique name for the UA instance, allowing it to obtain the same GRUU over time. It also provides an easy way to guarantee uniquess within the AOR. However, this specification does not require a long-lived and persistent instance identifier to properly function, and in some cases, there may be cause to use an identifier with weaker temporal persistence.

One URN that readily meets the requirements of this specification is the UUID URN [26], which allows for non-centralized computation of a URN based on time, unique names (such as a MAC address) or a random number generator. An example of a URN that would not meet the requirements of this specification is the national bibliographic number [16]. Since there is no clear relationship between an SIP UA instance and a URN in this namespace, there is no way a selection of a value can be performed that guarantees that another UA instance

doesn't choose the same value.

If a UA instance is registering against multiple AOR, it is RECOMMENDED that a UA instance provide a different contact URI for each AOR. This is needed for the UA to determine which GRUU to use as the remote target in responses to incoming dialog forming requests, as discussed in Section 8.1.

Besides the procedures discussed above, the REGISTER request is constructed identically to the case where this extension was not understood. Specifically, the contact in the REGISTER request SHOULD NOT contain the gruu Contact header field parameter, and the contact URI itself SHOULD NOT contain the grid parameter defined below. Any such parameters are ignored by the registrar, as the UA cannot propose a GRUU for usage with the contact.

If a UA wishes to guarantee that the request is not processed unless the domain supports and uses this extension, it MAY include a Require header field in the request with a value that contains the "gruu" option tag.

7.1.1.2 Processing the REGISTER Response

If the response is a 2xx, each Contact header field that contained the "+sip.instance" Contact header field parameter may also contain a "gruu" parameter. This parameter contains a SIP or SIPS URI that represents a GRUU corresponding to the UA instance that registered the contact. The URI will be a SIP URI if the To header field in the REGISTER request contained a SIP URI, else it will be a SIPS URI if the To header field in the REGISTER request contained a SIPS URI. Any requests sent to the GRUU URI will be routed by the domain to the contact with that instance ID. The GRUU will not normally change in subsequent 2xx responses to REGISTER. Indeed, even if the UA lets the contact expire, when it re-registers it at any later time, the registrar will normally provide the same GRUU for the same address-of-record and instance ID. However, as discussed above, this property cannot be completely guaranteed, as network failures may make it impossible to provide an identifier that persists for all time. As a result, a UA MUST be prepared to receive a different GRUU for the same instance ID/AOR pair in a subsequent registration response.

A non-2xx response to the REGISTER request has no impact on any existing GRUU previously provided to the UA. Specifically, if a previously successful REGISTER request provided the UA with a GRUU, a subsequent failed request does not remove, delete, or otherwise invalidate the GRUU.

7.1.2 Registrar Behavior

A registrar MAY create a GRUU for a particular instance ID/AOR pair at any time. Of course, if a UA requests a GRUU in a registration, and the registrar has not yet created one, it will need to do so in order to respond to the registration request. However, the registrar can create the GRUU in advance of any request from a UA.

A registrar MUST create both the SIP and SIPS versions of the GRUU, such that if the GRUU exists, both URI exist.

7.1.2.1 Processing a REGISTER Request

When a registrar compliant to this specification receives a REGISTER request, it checks for the presence of the Require header field in the request. If present, and if it contains the "gruu" option tag, the registrar MUST follow the procedures in the remainder of this section and Section 7.1.2.2 (that is, the procedures which result in the creation of new GRUUs for contacts indicating an instance ID, and the listing of GRUUs in the REGISTER response). If not present, but a Supported header field was present with the "gruu" option tag, the registrar SHOULD follow the procedures in the remainder of this section and Section 7.1.2.2. If the Supported header field was not present, or it if was present but did not contain the value "gruu", the registrar SHOULD NOT follow the procedures in the remainder of this section or Section 7.1.2.2.

As the registrar is processing the contacts in the REGISTER request according to the procedures of step 7 in Section 10.3 of RFC 3261, the registrar additionally checks whether each Contact header field in the REGISTER message contains a "+sip.instance" header field parameter. If present, the contact is processed further. If the registrar had not yet created a GRUU for that instance ID/AOR pair, it MUST do so at this time according to the procedures of Section 6. If the contact contained a "gruu" Contact header field parameter, it MUST be ignored by the registrar. A UA cannot suggest or otherwise provide a GRUU to the registrar.

Registration processing then continues as defined in RFC 3261. If, after that processing, that contact is bound to the AOR, it also becomes bound to the GRUU associated with that instance ID/AOR pair. If, after that processing, the contact was not bound to the AOR (due, for example, to an expires value of zero), the contact is not bound to the GRUU either. The registrar MUST store the instance ID along with the contact.

When generating the 200 (OK) response to the REGISTER request, the procedures of step 8 of Section 10.3 of RFC 3261 are followed.

Furthermore, for each Contact header field value placed in the response, if the registrar has stored an instance ID associated with that contact, that instance ID is returned as a Contact header field parameter, and furthermore, the server MUST add a "gruu" Contact header field parameter. The value of the gruu parameter is a quoted string containing the URI that is the GRUU for the associated instance ID/AOR pair. If the To header field in the REGISTER request had contained a SIP URI, the SIP version of the GRUU is returned. If the To header field in the REGISTER request had contained a SIPS URI, the SIPS version of the GRUU is returned.

The REGISTER response MUST contain a Require header field with the value "gruu". This is because the client needs to extract its GRUU from the REGISTER response, and utilize them as the remote target of dialog initiating requests and responses.

Note that handling of a REGISTER request containing a Contact header field with value "*" and an expiration of 0 still retains the meaning defined in RFC 3261 - all contacts, not just ones with a specific instance ID, are deleted. This removes their binding to the AOR and to any GRUU.

Inclusion of a GRUU in the "gruu" Contact header field parameter of a REGISTER response is separate from the computation and storage of the GRUU. It is possible that the registrar has computed a GRUU for one UA, but a different UA that queries for the current set of registrations doesn't understand GRUU. In that case, the REGISTER response sent to that second UA would not contain the "gruu" Contact header field parameter, even though the UA has a GRUU for that contact.

7.1.2.2 Timing Out a Registration

When a registered contact expires, its binding to the AOR is removed as normal. In addition, its binding to the GRUU is removed at the same time.

7.2 Administratively

Administrative creation of GRUUs is useful when a UA instance is a network server that is always available, and therefore doesn't register to the network. Examples of such servers are voicemail servers, application servers, and gateways.

There are no protocol operations required to administratively create a GRUU. The proxy serving the domain is configured with the GRUU, and with the contact it should be translated to. It is not strictly necessary to also configure the instance ID and AOR, since the

translation can be done directly. However, they serve as a useful tool for determining which resource and UA instance the GRUU is supposed to map to.

In addition to configuring the GRUU and its associated contact in the proxy serving the domain, the GRUU will also need to be configured into the UA instance associated with the GRUU.

It is also reasonable to model certain network servers as logically containing both a proxy and a UA instance. The proxy receives the request from the network, and passes it internally to the UA instance. In such a case, the GRUU routes directly to the server, and there is no need for a translation of the GRUU to a contact. The server itself would construct its own GRUU.

8. Using the GRUU

8.1 Sending a Message Containing a GRUU

A UA first obtains a GRUU using the procedures of Section 7, or by other means outside the scope of this specification.

A UA can use the GRUU in the same way it would use any other SIP or SIPS URI. However, a UA compliant to this specification MUST use a GRUU when populating the Contact header field of dialog-creating requests and responses. In other words, a UA compliant to this specification MUST use its GRUU as its remote target. This includes the INVITE request and its 2xx response, the SUBSCRIBE [6] request, its 2xx response, the NOTIFY request, and the REFER [7] request and its 2xx response.

If the UA instance has obtained multiple GRUUs (each for a different AOR) through a registration, it MUST use the one corresponding to the AOR used to send or receive the request. For sending a request, this means that the GRUU corresponds to the AOR present in the From header field, and furthermore the credentials used for authentication of the request correspond to the ones associated with that AOR. When receiving a request, the GRUU in the response corresponds to the AOR to which the original request was targeted. That AOR, however, will be rewritten by the proxy to correspond to the UA's registered contact. It is for this reason that different contacts are needed for each AOR that an instance registers against. In this way, when an incoming request arrives, the Request URI can be examined. It will be equal to a registered contact. That contact can be used to map directly to the AOR, and from there, the correct GRUU can be selected.

In those requests and responses where the GRUU is used as the remote

target, the UA MUST include a Supported header field that contains the option tag "gruu". However, it is not necessary for a UA to know whether or not its peer in the dialog supports this specification before using one as a remote target.

When using the GRUU as a remote target, a UA MAY add the "grid" URI parameter to the GRUU. This parameter MAY take on any value permitted by the grammar for the parameter. Note that there are no limitations on the size of this parameter. When a UA sends a request to the GRUU, the proxy for the domain that owns the GRUU will translate the GRUU in the Request-URI, replacing it with the URI bound to that GRUU. However, it will retain the "grid" parameter when this translation is performed. As a result, when the UA receives the request, the Request-URI will contain the "grid" created by the UA. This allows the UA to effectively manufacture an infinite supply of GRUU, each of which differs by the value of the "grid" parameter. When a UA receives a request that was sent to the GRUU, it will be able to tell which GRUU was invoked by the "grid" parameter.

An implication of this behavior is that all mid-dialog requests will be routed through intermediate proxies. There will never be direct, UA to UA signaling. It is anticipated that this limitation will be addressed in future specifications.

Once a UA knows that the remote target provided by its peer is a GRUU, it can use it in any application or SIP extension which requires a globally routable URI to operate. One such example is assisted call transfer.

8.2 Sending a Message to a GRUU

There is no new behavior associated with sending a request to a GRUU. A GRUU is a URI like any other. When a UA receives a request or response, it can know that the remote target is a GRUU if the request or response had a Supported header field that included the value "gruu". The UA can take the GRUU, and send a request to it, and then be sure that it is delivered to the UA instance which sent the request or response.

If the GRUU contains the "opaque" URI parameter, a UA can obtain the AOR for the user by stripping the parameter. The resulting URI is the AOR. If the GRUU does not have the "opaque" URI parameter, there is no mechanism defined for determining the AOR from the GRUU. Extraction of the AOR from the GRUU is useful for call logs and other accounting functions, where it is desirable to know the user to whom the request was directed.

Since the instance ID is a callee capabilities parameter, a UA might be tempted to send a request to the AOR of a user, and include an Accept-Contact header field [22] which indicates a preference for routing the request to a UA with a specific instance ID. Although this would appear to have the same effect as sending a request to the GRUU, it does not. The caller preferences expressed in the Accept-Contact header field are just preferences. Its efficacy depends on a UA constructing an Accept-Contact header field that interacts with domain processing logic for an AOR, to cause it to route to a particular instance. Given the variability in routing logic in a domain (for example, time based routing to only selected contacts), this doesn't work for many domain routing policies. However, this specification does not forbid a client from attempting such a request, as there may be cases where the desired operation truly is a preferential routing request.

8.3 Receiving a Request Sent to a GRUU

When a UAS receives a request sent to its GRUU, the incoming request URI will be equal to the contact that was registered (through REGISTER or some other action) by that UA instance. If the user agent had previously handed out its GRUU with a grid parameter, the incoming request URI may contain that parameter. This indicates to the UAS that the request is being received as a result of a request sent by the UAC to that GRUU/grid combination. This specification makes no normative statements about when to use a grid parameter, or what to do when receiving a request made to a GRUU/grid combination. Generally, any differing behaviors are a matter of local policy.

It is important to note that, when a user agent receives a request, and the request URI does not have a grid parameter, the user agent cannot tell whether the request was sent to the AOR or to the GRUU. As such, the UAS will process such requests identically. If a user agent needs to differentiate its behavior based on these cases, it will need to use a grid parameter.

8.4 Proxy Behavior

Proxy behavior is fully defined in Section 16 of RFC 3261. GRUU processing impacts that processing in two places - request targeting and record-routing.

8.4.1 Request Targeting

When a proxy server receives a request, and the proxy owns the domain in the Request URI, and the proxy is supposed to access a Location Service in order to compute request targets (as specified in Section 16.5 of RFC 3261 [1]), the proxy examines the Request URI. If the

Request URI is an AOR against which there are multiple registered contacts with the same instance ID parameter, the proxy MUST use only the most recently registered contact for inclusion in the target set. The contact that is the most recently registered is the one that has been bound to the AOR is the shortest period of time. This corresponds to the minimum value for the "duration-registered" attribute from the registration event package [27]. It is important to note that a refresh of the contact in a REGISTER message does not reset the duration it has been registered to zero. For example, if a softphone is started at 9am when a user logs into their computer, and the softphone refreshes its registration every hour, by 1230pm the contact has been registered for three and a half hours.

If the request URI is within the domain of the proxy, and the URI has been constructed by the domain such that the proxy is able to determine that it has the form of a GRUU for an AOR that is unknown within the domain, the proxy rejects the request with a 404. If the request URI is within the domain of the proxy, and the URI has been constructed by the domain such that the proxy is able to determine that it has the form of a GRUU for an AOR that known within the domain, but the instance ID is unknown, the proxy SHOULD generate a 480.

If the GRUU does exist, handling of the GRUU proceeds as specified in RFC 3261 Section 16. For GRUUs, the abstract location service described in Section 16.5 is utilized, producing a set of zero or more contacts, each of which is associated with the same instance ID. If there are more than one contacts bound to the GRUU, the proxy MUST select the one that has been most recently registered, as defined above. This produces zero or one contacts. The request target MUST be obtained by taking that one contact, and if the GRUU in the Request URI contained a "grid" URI parameter, adding that parameter to the request target. If the grid was already present in the contact bound to the GRUU, it is overwritten in this process. If no contacts were bound to the GRUU, the lookup of the GRUU in the abstract location service will result in zero target URI, eventually causing the proxy to reject the request with a 480 (Temorarily Unavailable) response.

If the contact had been registered using a Path header field [3], then that Path is used to construct the route set for reaching that contact through the GRUU as well as through the AOR, using the procedures specified in RFC 3327.

A proxy MAY apply other processing to the request, such as execution of called party features, as discussed in Section 6.

A request sent to a GRUU SHOULD NOT be redirected. In many

instances, a GRUU is used by a UA in order to assist in the traversal of NATs and firewalls, and a redirection may prevent such a case from working.

8.4.2 Record Routing

As described above, a user agent uses its GRUU as a remote target. This has an impact on the path taken by subsequent mid-dialog requests. Depending on the desires of the proxies involved, this may impact record route processing.

Two cases can be considered. The first is shown in Figure 3. In this case, there is a single proxy in the user's domain. An incoming INVITE request arrives for the users AOR (1) and is forwarded to the user agent at its registered contact C1 (2). The proxy inserts a Record-Route header field into the proxied request, with a value of R1. The user agent generates a 200 OK to the request, using its GRUU G1 as the remote target.

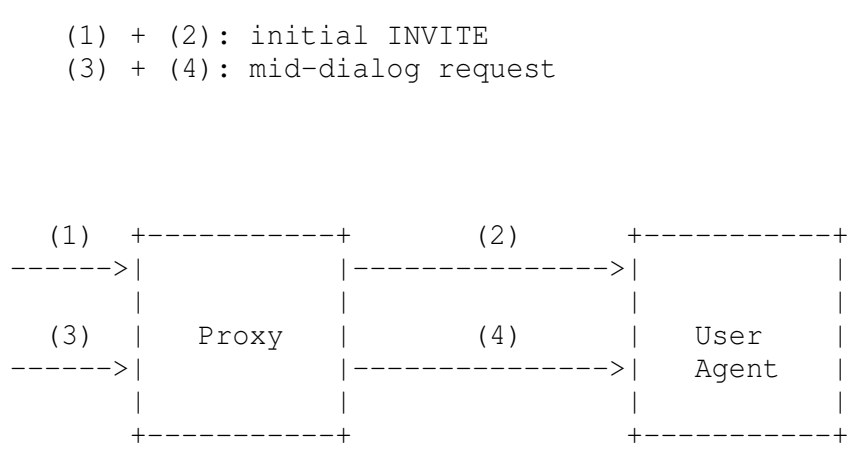


Figure 3

When a mid-dialog request shows up destined for the user agent (message 3), it will arrive at the proxy in the following form:

```
INVITE G1
Route: R1
```

Since the top Route header field value identifies the proxy, the proxy removes it. As there are no more Route header field values, the proxy processes the request URI. However, the request URI is a GRUU, and is therefore a domain under the control of the proxy. The proxy will need to perform the processing of Section 8.4.1, which

will result in the translation of the GRUU into the contact C1, followed by transmission of the request to the user agent (message 4).

This sequence of processing in the proxy is somewhat unusual, in that mid-dialog requests (that is, requests with a Route header field that a proxy inserted as a result of a Record-Route operation) do not normally cause a proxy to have to invoke a location service to process the request URI. It is for this reason that this is called out here.

The previous case assumed that there was a single proxy in the domain. In more complicated cases, there can be two or more proxies within a domain on the initial request path. This is shown in Figure 5. In this figure, there is a home proxy, to which requests targeted to the AOR are sent. The home proxy executes the abstract location service and runs user features. The edge proxy acts as the outbound proxy for users, performs authentication, manages TCP/TLS connections to the client, and does other functions associated with the transition from the provider proxy network to the client. This specific division of responsibilities between home and edge proxy is just for the purposes of illustration; the discussion applies to a disaggregation of proxy logic into any number of proxies. In such a configuration, registrations from the user agent would pass through the edge proxy, which would insert a Path header field [3] for itself.

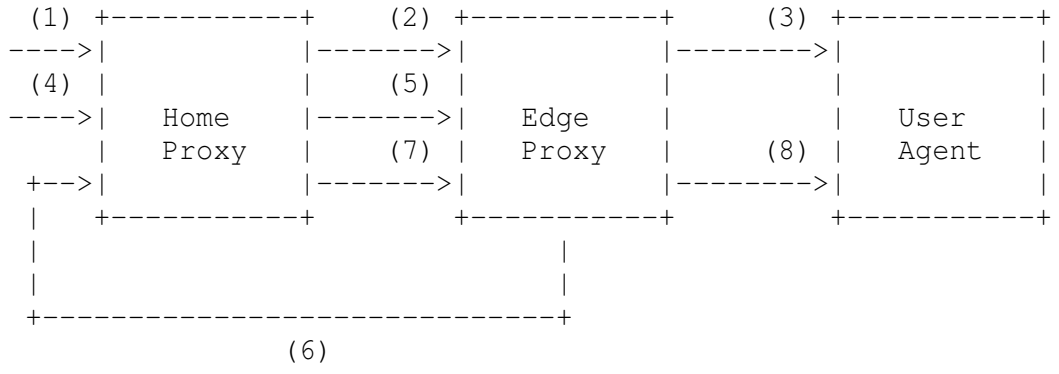


Figure 5

When an incoming request arrives for the AOR (message 1), the home proxy would look it up, discover the registered contact and Path, and then send the request to the edge proxy as a result of the Route header field inserted with the Path value. The home proxy record routes with the URI H1. The edge proxy would forward the request to the request URI (which points to the client), and insert a Record-Route header field value with the URI E1 (message 2). This request is accepted by the user agent, which inserts its GRUU G1 as the remote target.

When the peer in the dialog sends a mid-dialog request, it will have the following form:

```
INVITE G1
Route: H1, E1
```

This request will arrive at the home proxy (due to H1 in the Route header field) (message 4). The home proxy will forward it to the edge proxy (due to E1 in the Route header field) (message 5). The edge proxy, seeing no more Route header field values, sends the request to the Request URI. This is a GRUU, and like an AOR, will route to the home proxy. This causes the request to loop back around (message 6). The home proxy performs the GRUU processing of Section 8.4.1, causing the request to be forwarded to the edge proxy a second time (this time, as a result of a Route header field value obtained from the Path header in the registration) (message 7), and then delivered to the client (message 8).

While this flow works, it is highly inefficient, as it causes each mid-dialog request to spiral route. If this behavior is not desirable. To prevent it, the following procedures SHOULD be followed. When a client generates a REGISTER request, this request passes through the edge proxy on its way to the home proxy. The REGISTER request will contain the AOR of the user (in the To header field) and also indicate whether or not the GRUU extension is supported. The proxy can decide to insert itself on the Path on a case by case basis. However, if it does so for one registration, it SHOULD do so for all registrations for the same AOR. The value of the Path header field inserted by the proxy SHOULD be constructed so that it indicates whether or not the proxy inserted itself on the Path for this AOR.

When a request arrives from the home proxy towards the client, the proxy inspects the Route header field. This header field will contain the URI the edge proxy had placed into the Path. If the value indicates that the edge proxy had put itself on the Path for the registration from this client, there is no need for the proxy to

retain its record-route in the response. The proxy MAY remove its record-route value from the 200 OK response in this case. If the value indicates that the proxy had not put itself on the Path, it would retain the Record-Route in the response.

Similarly, if a request arrives from the client towards the home proxy, the edge proxy would look at the identity of the sender of the request. If the proxy knows that it is placing itself on the Path for registrations from that AOR, the edge proxy would insert a Record-Route into the request, and then remove it in the response. Similarly, if the identity of the sender of the request is one for which the client has not put itself on the Path, the edge proxy would keep its Record-Route in the response.

Removing its Record-Route value from the response will result in a different route set as seen by the caller and callee; the callee (which is the user agent in the figure) will have a route set entry for its edge proxy, while the caller will not. The caller will have a route set entry for its edge proxy, while the callee will not.

In such a case, a mid-dialog request that arrives at the home proxy will be of the form:

```
INVITE G1
Route: H1
```

This does the "right thing" and causes the request to be routed from the home proxy to the edge proxy to the client, without the additional spiral.

9. The opaque SIP URI Parameter

This specification defines a new SIP URI parameter, "opaque". This parameter is useful for constructing GRUUs, but is a generally valuable tool for building URI that are linked to another URI in some way.

The "opaque" parameter has no explicit semantics. It is merely a repository of information whose interpretation is at the discretion of the entity that creates the URI. This means that an element that constructs a URI with the "opaque" parameter MUST ensure that it routes back to itself or another element that can interpret the content of the parameter. The "opaque" parameter can be viewed as a form of cookie for this reason.

If the "opaque" parameter in the URI is removed, the resulting URI MUST correspond to a valid resource in the domain to which the URI

with the "opaque" parameter is associated. The nature of the association is determined from the context in which the URI was obtained. When used to construct a GRUU, it means that the URI formed by stripping the "opaque" parameter MUST correspond to the AOR associated with the GRUU. The recipient of a GRUU cannot determine that it is a GRUU by direct examination of the URI. However, the recipient may know if it received the GRUU in the Contact header field of a SIP request or response that contained a Supported header field with the option tag "gruu". If it knows its a GRUU through such context, and the GRUU contains the "opaque" parameter, the UA knows it can obtain the AOR by removing the "opaque" parameter.

Other possible uses of the "opaque" URI parameter include constructing of service URIs for a user, such as their voicemail inbox or personal conference bridge.

10. Grammar

This specification defines two new Contact header field parameters, gruu and +sip.instance, and two new URI parameters, "grid" and "opaque". The grammar for string-value is obtained from [11], and the grammar for uric is defined in RFC 3986 [9].

```
contact-params    =  c-p-q / c-p-expires / c-p-gruu / cp-instance
                      / contact-extension
c-p-gruu          =  "gruu" EQUAL DQUOTE (SIP-URI / SIPS-URI) DQUOTE
cp-instance       =  "+sip.instance" EQUAL LDQUOT "<"
                     instance-val ">" RDQUOT
uri-parameter     =  transport-param / user-param / method-param
                     / ttl-param / maddr-param / lr-param / grid-param
                     / opaque-param / other-param
grid-param        = "grid=" pvalue        ; defined in RFC3261
opaque-param      = "opaque=" pvalue      ; defined in RFC3261
instance-val      = *uric ; defined in RFC 2396
```

11. Requirements

This specification was created in order to meet the following requirements:

REQ 1: When a UA invokes a GRUU, it MUST cause the request to be routed to the specific UA instance to which the GRUU refers.

REQ 2: It MUST be possible for a GRUU to be invoked from anywhere on the Internet, and still cause the request to be routed appropriately. That is, a GRUU MUST NOT be restricted to use within a specific addressing realm.

REQ 3: It MUST be possible for a GRUU to be constructed without requiring the network to store additional state.

REQ 4: It MUST be possible for a UA to obtain a multiplicity of GRUUs, each one of which routes to that UA instance. This is needed to support ad-hoc conferencing, for example, where a UA instance needs a different URI for each conference it is hosting.

REQ 5: When a UA receives a request sent to a GRUU, it MUST be possible for the UA to know the GRUU which was used to invoke the request. This is necessary as a consequence of requirement 4.

REQ 6: It MUST be possible for a UA to add opaque content to a GRUU, which is not interpreted or altered by the network, and used only by the UA instance to whom the GRUU refers. This provides a basic cookie type of functionality, allowing a UA to build a GRUU with state embedded within it.

REQ 7: It MUST be possible for a proxy to execute services and features on behalf of a UA instance represented by a GRUU. As an example, if a user has call blocking features, a proxy may want to apply those call blocking features to calls made to the GRUU in addition to calls made to the user's AOR.

REQ 8: It MUST be possible for a UA in a dialog to inform its peer of its GRUU, and for the peer to know that the URI represents a GRUU. This is needed for the conferencing and dialog reuse applications of GRUUs, where the URIs are transferred within a dialog.

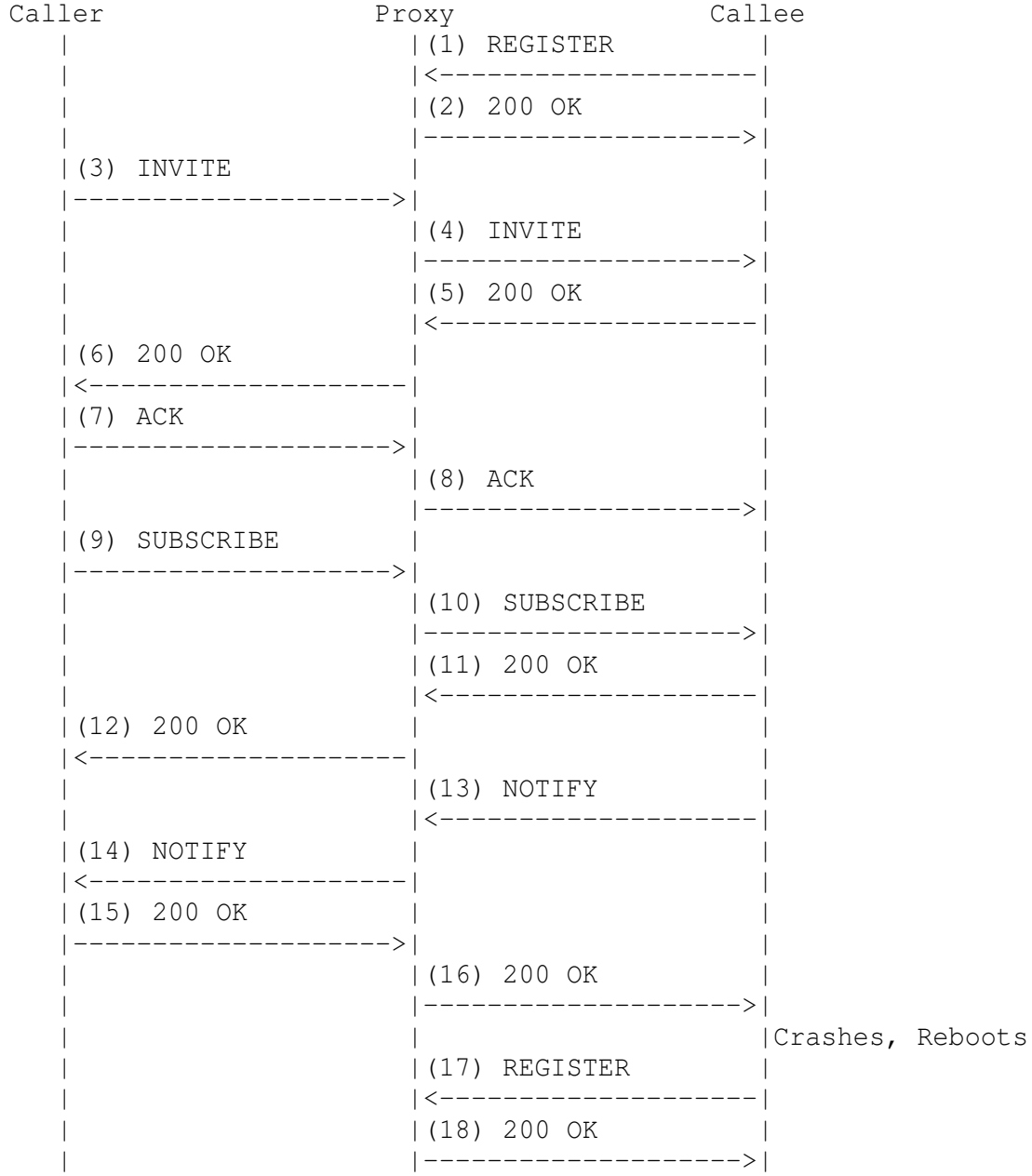
REQ 9: When transferring a GRUU per requirement 8, it MUST be possible for the UA receiving the GRUU to be assured of its integrity and authenticity.

REQ 10: It MUST be possible for a server, authoritative for a domain, to construct a GRUU which routes to a UA instance bound to an AOR in that domain. In other words, the proxy can construct a GRUU too. This is needed for the presence application.

12. Example Call Flow

The following call flow shows a basic registration and call setup, followed by a subscription directed to the GRUU. It then shows a

failure of the callee, followed by a re-registration. The conventions of [21] are used to describe representation of long message lines.



The Callee supports the GRUU extension. As such, its REGISTER (1) looks like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=a73kszlfl
Supported: gruu
To: Callee <sip:callee@example.com>
Call-ID: 1j9FpLxk3uxtm8tn@192.0.2.1
CSeq: 1 REGISTER
Contact: <sip:callee@192.0.2.1>
  ;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
Content-Length: 0
```

The REGISTER response would look like:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnashds7
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com> ;tag=b88sn
Require: gruu
Call-ID: 1j9FpLxk3uxtm8tn@192.0.2.1
CSeq: 1 REGISTER
<allOneLine>
Contact: <sip:callee@192.0.2.1>
 ;gruu="sip:callee@example.com;
opaque=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
;expires=3600
</allOneLine>
Content-Length: 0
```

Note how the Contact header field in the REGISTER response contains the gruu parameter with the URI sip:callee@example.com;opaque=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6. This represents a GRUU that translates to the contact sip:callee@192.0.2.1.

The INVITE from the caller is a normal SIP INVITE. The 200 OK generated by the callee (message 5), however, now contains a GRUU as the remote target. The UA has also chosen to include a grid URI parameter into the GRUU.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bKnaa8
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK99a
From: Caller <sip:caller@example.com>;tag=n88ah
To: Callee <sip:callee@example.com> ;tag=a0z8
Call-ID: 1j9FpLxk3uxtma7@host.example.com
CSeq: 1 INVITE
Supported: gruu
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK
<allOneLine>
Contact:
<sip:callee@example.com
;opaque=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6;grid=99a>
</allOneLine>
Content-Length: --
Content-Type: application/sdp

[SDP Not shown]
```

At some point later in the call, the caller decides to subscribe to the dialog event package [20] at that specific UA. To do that, it generates a SUBSCRIBE request (message 9), but directs it towards the remote target, which is a GRUU:

```
<allOneLine>
SUBSCRIBE sip:callee@example.com;opaque=urn:uuid:f8
1d4fae-7dec-11d0-a765-00a0c91e6bf6;grid=99a
 SIP/2.0
</allOneLine>
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:caller@example.com>;tag=kkaz-
To: Callee <sip:callee@example.com>
Call-ID: faif9a@host.example.com
CSeq: 2 SUBSCRIBE
Supported: gruu
Event: dialog
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK
Contact: <sip:caller@example.com;opaque=hdg7777ad7aflzig8sf7>
Content-Length: 0
```

In this example, the caller itself supports the GRUU extension, and is using its own GRUU to populate its remote target.

This request is routed to the proxy, which proceeds to perform a location lookup on the request URI. It is translated into the contact for that instance, and then proxied there (message 10 below). Note how the grid parameter is maintained.

```
SUBSCRIBE sip:callee@192.0.2.1;grid=99a SIP/2.0
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bK9555
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:caller@example.com>;tag=kkaz-
To: Callee <sip:callee@example.com>
Call-ID: faif9a@host.example.com
CSeq: 2 SUBSCRIBE
Supported: gruu
Event: dialog
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK
Contact: <sip:caller@example.com;opaque=hdg7777ad7aflzig8sf7>
Content-Length: 0
```

At some point after message 16 is received, the callee's machine crashes and recovers. It obtains a new IP address, 192.0.2.2. Unaware that it had previously had an active registration, it creates a new one (message 17 below). Notice how the instance ID remains the same, as it persists across reboot cycles:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.2;branch=z9hG4bKnasbba
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=ha8d777f0
Supported: gruu
To: Callee <sip:callee@example.com>
Call-ID: hf8asxzff8s7f@192.0.2.2
CSeq: 1 REGISTER
Contact: <sip:callee@192.0.2.2>
  ;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
Content-Length: 0
```

The registrar notices that a different contact, sip:callee@192.0.2.1, is already associated with the same instance ID. It registers the new one too and returns both in the REGISTER response. Both have the same GRUU. However, only this new contact (the most recently registered one) will be used by the proxy for population in the target set. It then generates the following response:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.2;branch=z9hG4bKnasbba
From: Callee <sip:callee@example.com>;tag=ha8d777f0
To: Callee <sip:callee@example.com>;tag=99f8f7
Require: gruu
Call-ID: hf8asxzff8s7f@192.0.2.2
CSeq: 1 REGISTER
<allOneLine>
Contact: <sip:callee@192.0.2.2>
;gruu="sip:callee@example.com;opaque=urn:
uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
;expires=3600
</allOneLine>
Contact: <sip:callee@192.0.2.1>
;gruu="sip:callee@example.com;opaque=urn:
uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
;expires=400
</allOneLine>
Content-Length: 0
```

13. Security Considerations

GRUUs do not provide a solution for privacy. In particular, since the GRUU does not change during the lifetime of a registration, an attacker could correlate two calls as coming from the same source, which in and of itself reveals information about the caller. Furthermore, GRUUs do not address other aspects of privacy, such as the addresses used for media transport. For a discussion of how privacy services are provided in SIP, see RFC 3323 [14].

It is important for a UA to be assured of the integrity of a GRUU when it is given one in a REGISTER response. If the GRUU is tampered with by an attacker, the result could be denial of service to the UA. As a result, it is RECOMMENDED that a UA use the SIPS URI scheme in the Request-URI when registering.

The example GRUU construction algorithm in Appendix A.1 makes no attempt to create a GRUU that hides the AOR and instance ID associated with the GRUU. In general, determination of the AOR associated with a GRUU is considered a good property, since it allows for easy tracking of the target of a particular call. Learning the instance ID provides little benefit to an attacker. To register or otherwise impact registrations for the user, an attacker would need to obtain the credentials for the user. Knowing the instance ID is insufficient.

The example GRUU construction algorithm in Appendix A.1 makes no attempt to create a GRUU that prevents users from guessing a GRUU based on knowledge of the AOR and instance ID. A user that is able to do that will be able to direct a new request at a particular instance. However, this specification recommends that service treatment be given to requests that are sent to a GRUU, including screening features in particular. That treatment will make sure that the GRUU does not provide a back door for attackers to contact a user that has tried to block the attacker.

14. IANA Considerations

This specification defines a new Contact header field parameter, two SIP URI parameters, a media feature tag and a SIP option tag.

14.1 Header Field Parameter

This specification defines a new header field parameter, as per the registry created by [12]. The required information is as follows:

Header field in which the parameter can appear: Contact

Name of the Parameter gruu

RFC Reference RFC XXXX [[NOTE TO IANA: Please replace XXXX with the RFC number of this specification.]]

14.2 URI Parameters

This specification defines two new SIP URI parameters, as per the registry created by [13].

Name of the Parameter grid

RFC Reference RFC XXXX [[NOTE TO IANA: Please replace XXXX with the RFC number of this specification.]]

Name of the Parameter opaque

RFC Reference RFC XXXX [[NOTE TO IANA: Please replace XXXX with the RFC number of this specification.]]

14.3 Media Feature Tag

This section registers a new media feature tag, per the procedures defined in RFC 2506 [8]. The tag is placed into the sip tree, which

is defined in [11].

Media feature tag name: sip.instance

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag contains a string containing a URI, and ideally a URN, that indicates a unique identifier associated with the UA instance registering the Contact.

Values appropriate for use with this feature tag: String.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a specific device.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

Security Considerations: This media feature tag can be used in ways which affect application behaviors. For example, the SIP caller preferences extension [22] allows for call routing decisions to be based on the values of these parameters. Therefore, if an attacker can modify the values of this tag, they may be able to affect the behavior of applications. As a result of this, applications which utilize this media feature tag SHOULD provide a means for ensuring its integrity. Similarly, this feature tag should only be trusted as valid when it comes from the user or user agent described by the tag. As a result, protocols for conveying this feature tag SHOULD provide a mechanism for guaranteeing authenticity.

14.4 SIP Option Tag

This specification registers a new SIP option tag, as per the guidelines in Section 27.1 of RFC 3261.

Name: gruu

Description: This option tag is used to identify the Globally Routable User Agent URI (GRUU) extension. When used in a Supported header, it indicates that a User Agent understands the extension, and has included a GRUU in the Contact header field of

its dialog initiating requests and responses. When used in a Require header field of a REGISTER request, it indicates that the registrar should assign a GRUU to the Contact URI.

15. Acknowledgements

The author would like to thank Rohan Mahy, Paul Kyzivat, Alan Johnston, and Cullen Jennings for their contributions to this work.

16. References

16.1 Normative References

[1] 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.

[2] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[3] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[4] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

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

[6] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[7] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[8] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, March 1999.

[9] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[10] Moats, R., "URN Syntax", RFC 2141, May 1997.

[11] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol

(SIP)", RFC 3840, August 2004.

[12] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.

[13] Camarillo, G., "The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP)", BCP 99, RFC 3969, December 2004.

16.2 Informative References

[14] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[15] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003.

[16] Hakala, J., "Using National Bibliography Numbers as Uniform Resource Names", RFC 3188, October 2001.

[17] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-05 (work in progress), May 2005.

[18] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", draft-ietf-sip-outbound-00 (work in progress), July 2005.

[19] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[20] Rosenberg, J., "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-06 (work in progress), April 2005.

[21] Sparks, R., "Session Initiation Protocol Torture Test Messages", draft-ietf-sipping-torture-tests-07 (work in progress), May 2005.

[22] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[23] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.

[24] Sparks, R. and A. Johnston, "Session Initiation Protocol Call Control - Transfer", draft-ietf-sipping-cc-transfer-04 (work in progress), April 2005.

[25] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[26] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[27] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

Author's Address

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Appendix A. Example GRUU Construction Algorithms

The mechanism for constructing a GRUU is not subject to specification. This appendix provides two examples that can be used by a registar. Others are, of course, permitted, as long as they meet the constraints defined for a GRUU.

A.1 Instance ID in opaque URI Parameter

The most basic approach for constructing a GRUU is to utilize the "opaque" URI parameter. The user and domain portions of the URI are equal to the AOR, and the "opaque" parameter is populated with the instance ID.

A.2 Encrypted Instance ID and AOR

In many cases, it will be desirable to construct the GRUU in such a way that it will not be possible, based on inspection of the URI, to

determine the Contact URI that the GRUU translates to. It may also be desirable to construct it so that it will not be possible to determine the instance ID/AOR pair associated with the GRUU. Whether or not a GRUU should be constructed with this property is a local policy decision.

With these rules, it is possible to construct a GRUU without requiring the maintenance of any additional state. To do that, the URI would be constructed in the following fashion:

```
user-part = "GRUU" | BASE64(E(K, (salt | " " | AOR | " " |
instance ID)))
```

Where E(K,X) represents a suitable encryption function (such as AES with 128 bit keys) with key K applied to data block X, and the "|" operator implies concatenation. The single space (" ") between components is used as a delimeter, so that the components can easily be extracted after decryption. Salt represents a random string that prevents a client from obtaining pairs of known plaintext and ciphertext. A good choice would be at least 128 bits of randomness in the salt.

This mechanism uses the user-part of the SIP URI to convey the encrypted AOR and instance ID. The user-part is used instead of the "opaque" URI parameter because of the desired anonymity properties.

The benefit of this mechanism is that a server need not store additional information on mapping a GRUU to its corresponding contact. The user part of the GRUU contains the instance ID and AOR. Assuming that the domain stores registrations in a database indexed by the AOR, the proxy processing the GRUU would look up the AOR, extract the currently registered contacts, and find the one matching the instance ID encoded in the request URI. The contact whose instance ID is that instance ID is then used as the translated version of the GRUU. Encryption is needed to prevent attacks whereby the server is sent requests with faked GRUU, causing the server to direct requests to any named URI. Even with encryption, the proxy should validate the user part after decryption. In particular, the AOR should be managed by the proxy in that domain. Should a UA send a request with a fake GRUU, the proxy would decrypt and then discard it because there would be no URI or an invalid URI inside.

While this approach has many benefits, it has the drawback of producing fairly long GRUUs.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP WG
Internet-Draft
Expires: January 12, 2006

C. Jennings, Ed.
Cisco Systems
R. Mahy, Ed.
SIP Edge LLC
July 11, 2005

Managing Client Initiated Connections in the Session Initiation Protocol (SIP)
draft-ietf-sip-outbound-00

Status of this Memo

Copyright Notice

Abstract

Session Initiation Protocol (SIP) allows proxy servers to initiate TCP connections and send asynchronous UDP datagrams to User Agents in order to deliver requests. However, many practical considerations, such as the existence of firewalls and NATs, prevent servers from connecting to User Agents in this way. Even when a proxy server can open a TCP connection to a User Agent, most User Agents lack a

certificate suitable to act as a TLS server. This specification defines behaviors for user agents, registrars and proxy servers that allow requests to be delivered on existing connections established by the User Agent. It also defines keep alive behaviors needed to keep NAT bindings open and specifies the usage of multiple connections for high availability systems.

Table of Contents

1. Introduction

There are many environments for SIP deployments in which the User Agent (UA) can form a connection to a Registrar or Proxy but in which the connections in the reverse direction to the UA are not possible. This can happen for several reasons. Connection to the UA can be blocked by a firewall device between the UA and the proxy or registrar, which will only allow new connections in the direction of the UA to the Proxy. Similarly there may be a NAT, which are only capable of allowing new connections from the private address side to the public side. It is worth noting that most UAs in the world are deployed behind firewalls or NATs.

Most IP phones and personal computers get their network configurations dynamically via a protocol such as DHCP. These systems typically do not have a useful name in DNS, and they definitely do not have a long-term, stable DNS name that is appropriate for binding to a certificate. It is impractical for them to have a certificate that can be used as a client-side TLS certificate for SIP. However, these systems can still form TLS connections to a proxy or registrar such that the UA authenticates the server certificate, and the server authenticates the UA using a shared secret in a digest challenge.

The key idea of this specification is that when a UA sends a REGISTER request, the proxy can later use this same connection to forward any requests that need to go to this UA. For a UA to receive incoming requests, the UA has to connect to the server. Since the server can't connect to the UA, the UA has to make sure that a connection is always active. This requires the UA to detect when a connection fails. Since, such detection takes time and leaves a window of opportunity for missed incoming requests, this mechanism allows the UA to use multiple connections, referred to as "flows", to the proxy or registrar and using a keep alive mechanism on each flow so that the UA can detect when a flow has failed.

2. Conventions and 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 [2].

2.1 Definitions

Edge Proxy: An Edge Proxy is any proxy that is located topologically between the registering user agent and the registrar.

flow: A Flow is a network protocol layer connection between two hosts that is represented by the network address of both ends and the protocol. For TCP and UDP this would include the IP addresses and ports of both ends and the protocol (TCP or UDP). With TCP, a flow would often have to one to one correspondence with a single file descriptor in the operating system.

flow-id: This refers to the value of a new header parameter value for the contact header. When UA register multiple times, each registration gets a unique flow-id value.

instance-id: This specification uses the word instance-id to refer to the value of the "sip.instance" media feature tag in the Contact header field. This is a URN that uniquely identifies the UA.

3. Overview

Several scenarios in which this technique is useful are discussed below, including the simple collocated registrar and proxy, a user agent desiring multiple connections to a resource (for redundancy for example), and an system that uses Edge Proxies.

3.1 Summary of Mechanism

The overall approach is fairly simple. Each UA has a unique instance-id that stays the same for this UA even if the UA reboots or is power cycled. Each UA can register multiple times. Each registration includes the instance-id for the UA and a flow-id label that is different for each connection.

UAs use a keep alive mechanism to keep their flow to the proxy or registrar alive. For TCP, TLS, and other connection oriented protocols this is a burst containing a single CRLF. For UDP it is a STUN request sent over the flow. A UA can create more than one flow using multiple registrations for the same AOR. The instance-id parameter is used by the proxy to identify with which UA a flow is associated. The flow-id is used by the proxy and registrar to tell the difference between a UA re-registering and one that is registering over an additional flow. The proxies keep track of the flows used for successful registrations.

When a proxy goes to route a message to a UA for which it has a binding, it can use any one of the flows on which a successful registration has been completed. A failure on a particular flow can be tried again on an alternate flow. Proxies can determine which flows go to the same UA by looking at the instance-id. Proxies can tell that a flow replaces a previous abandoned flow by looking at the flow-id.

3.2 Single Registrar and UA

In this example there is single server acting as both a registrar and proxy.

User Agents forming only a single connection continue to register normally but include the instance-id as described in the GRUU [1] specification and can also add a flow-id parameter to the Contact header field value. The flow-id parameter is used to allow the registrar to detect and avoid using invalid contacts when a UA reboots, as described later in this section.

For clarity, here is an example. Bob's UA creates a new TCP flow to the registrar and sends the following REGISTER request.

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bK-bad0ce-11-1036
Max-Forwards: 70
From: Bob <sip:bob@example.com>;tag=d879h76
To: Bob <sip:bob@example.com>
Call-ID: 8921348ju72je840.204
CSeq: 1 REGISTER
Contact: <sip:line1@192.168.0.2>; flow-id=1;
  ;+sip.instance="<urn:uuid:00000000-0000-0000-0000-000A95A0E128>"
Content-Length: 0
```

Implementors often ask why the value of the sip.instance is inside angle brackets. This is a requirement of RFC 3840 [8] which defines that media feature tags in SIP. Feature tags which are strings are compared by case sensitive string comparison. To differentiate these tags from tokens (which are not case sensitive), case sensitive parameters such as the sip.instance media feature tag are placed inside angle brackets.

The registrar challenges this registration to authenticate Bob. When the registrar adds an entry for this contact under the AOR for Bob,

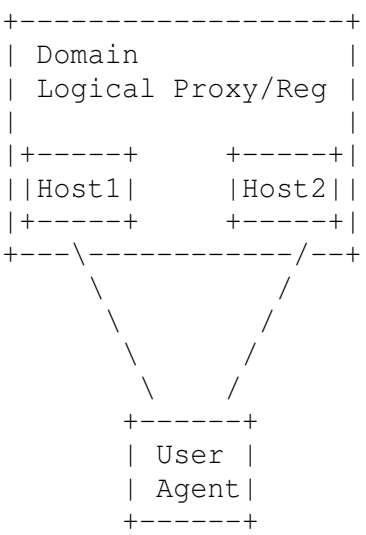
the registrar also keeps track of the connection over which it received this registration.

The registrar saves the instance-id (as defined in [1]) and flow-id (as defined in Section 9) along with the rest of the Contact header. If the instance-id and flow-id are the same as a previous registration for the same AOR, the proxy uses the most recently created registration first. This allows a UA that has rebooted to replace its previous registration for each flow with minimal impact on overall system load.

Later when Alice sends a request to Bob, his proxy selects target set. The proxy forwards the request to elements in the target set based on the proxies policy. The proxy looks at the the target set and uses the instance-id to understand that two targets both end up routing to the same UA. When the proxy goes for forward a request to a given target, it looks and finds the flows that received this registrations. The proxy then forwards the request on that flow instead of trying to form a new flow to that contact. This allows the proxy to forward a request to a particular contact down the same flow that did the registration for this AOR. If the proxy had multiple flows that all went to this UA, it could choose any one of registration binding that it had for this AOR and had the same instance-id as the selected UA. In general, if two registrations have the same flow-id and instance-id, the proxy would favor the most recently registered flow. This is so that if a UA reboots, the proxy will prefer to use the most recent flow that goes to this UA instead of trying one of the old flows which will presumably fail.

3.3 Multiple Connections from a User Agent

In this example system, the logical proxy/registrar for the domain is running on two hosts that share the appropriate state and can both provide registrar and proxy functionality for the domain. The UA will form connections to two of the physical hosts for the domain.



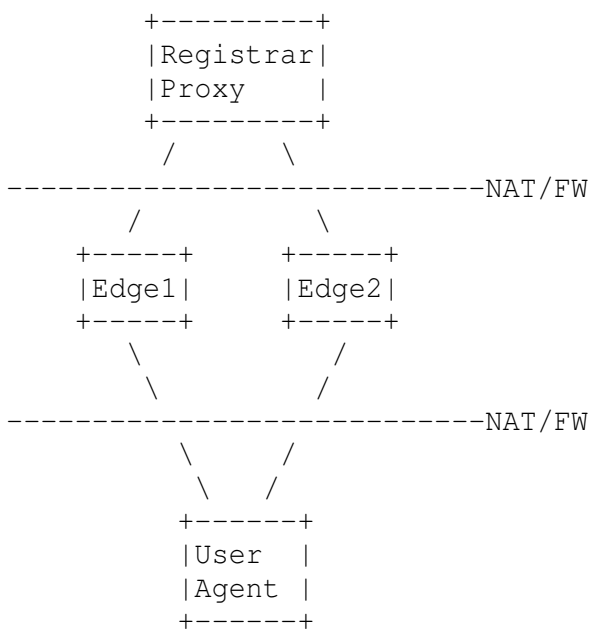
The UA is configured with a primary and backup registration URI. The administrative domain that created these URIs MUST insure that the two URIs resolve to separate hosts. These URI have normal SIP processing so things like SRV can be used to do load balance across a proxy farm.

The proxies can all use the Path header (as described in the next section) to insure that a route to each connection is available to each host, or the logical proxy can implement its own mechanism.

When a single server fails, all the UAs that have a registration with it will detect this and try and reconnect. This can cause large loads on the server and is referred to as the avalanche restart problem. The multiple flows to many servers help reduce the load caused by the avalanche restart. If a UA has multiple flows, and one os the servers fails, it can delay some significant time before trying to form a new connection to replace the flow to the server that failed. By spreading out the time used for all the UA to reconnect to a server, the load on the server is reduced.

3.4 Edge Proxies

Some SIP deployments use edge proxies such that the UA sends the REGISTER to an edge proxy that then forwards the REGISTER to the Registrar. The edge proxy includes a Path header [11] so that when the registrar later forwards a request to this UA, the request is routed through the edge proxy. There could be a NAT for FW between the UA and the edge proxy and there could also be one between the edge proxy and the Registrar. This second case typically happens when the Edge proxy is in an enterprise the the registrar is at a service provider.



These systems can use effectively the same mechanism as described in the previous sections but need to use the Path header. When the edge proxy receives a registration, it needs to create an identifier value that is unique to this flow (and not a subsequent flow with the same addresses) and put this identifier in the path header. This is done by putting the value in the user portion of a loose route in the path header. If the registration succeeds, the edge proxy needs to map future requests that are routed to the identifier value that was put in the Path header to the associated flow.

3.5 Keep Alive Techniques

A keep alive mechanism needs to detect both failure of a connection and changes to the NAT public mapping. When a residential NAT is rebooted, the UA needs to understand that its bindings are no longer valid and it needs to re-register. Simply sending keep alive packets will not detect this failure when using UDP. With connection oriented transports such as TCP or TLS, the keep alive will detect failure after a NAT reboot. Connection oriented transport failures are detected by having the UA periodically sends a CRLF over the connection; if the connection has failed, a connection level error will be reported to the UA. A CRLF can be considered the beginning of the next message that will be sent, and therefore this approach is backwards compatible with the core SIP specification.

> Note: The TCP KEEP_ALIVE mechanism is not used because most operating systems do not allow the time to be set on a per connection basis. Linux, Solaris, OS X, and Windows all allow

KEEP_ALIVEs to be turned on or off on a single socket using the SO_KEEPALIVE socket options but can not change the duration of the timer for an individual socket. The length of the timer typically defaults to 7200 seconds. The length of the timer can be changed to a smaller value by setting a kernel parameter but that affects all TCP connections on the host and thus is not appropriate to use.

The keep alive mechanism for UDP is quite different. The UA needs to detect when the connection is working but also when the flow definition has changed. A flow definition could change because a NAT device in the network path reboots and the resulting public IP address or port mapping for the UA changes. To detect this, STUN [5] requests are sent over the connection that is being used for the UDP SIP traffic. The proxy or registrar acts as a STUN server on the SIP signaling port.

Note: The STUN mechanism is very robust and allows the detection of a changed IP address. It may also be possible to do this with OPTIONS messages and rport; although this approach has the advantage of being backwards compatible, it also increases the load on the proxy or registrar server.

If the UA detects that the connection has failed or that the flow definition has changed, it needs to re-register using a back-off mechanism described in Section 4 in order to provide congestion relief when a large number of agents simultaneously reboot.

4. User Agent Mechanisms

The UA behavior is divided up into sections. The first describes what a client must do when forming a new connection, the second when detecting failure of a connection, and the third on failure recovery.

4.1 Forming Flows

UAs are configured one of more SIP URIs with which to register. A UA MUST support sets with at least two URIs (primary and backup) and SHOULD support sets with up to four URIs. For each URI in the redundancy set, the UA MUST send a REGISTER with a loose route set to the URI from the set. The UA MUST include the the instance-id as described in the [1]. The UA MUST also add a distinct flow-id parameter to the contact header. The UA SHOULD use a flow-id value of 1 for the first URI in the set, and a flow-id value of 2 for the second, and so on. Each one of these registrations will form a new flow from the UA to the proxy.

Note that the UA needs to honor 503 responses to registrations as

described in RFC 3261 and RFC 3263. In particular implementers should note that a 503 with a Retry-After is not considered a failure to form the connection. The UA should wait the indicated amount of time and retry the connection. A Retry-After header field value of 0 is valid and indicates the UA should retry the REGISTER immediately. Implementations need to ensure that when retrying the REGISTER they redo the DNS resolution process such that if multiple hosts are reachable from the URI, there is a chance that the UA will select an alternate host from the one it chose the previous time the URI was resolved.

4.1.1 Instance-ID Selection

The instance-id needs to be a URN but there are many ways one can be generated. A particularly simple way for both "hard" phones and "soft" phones is to use a UUID as defined in [7]. A device like a soft-phone, when first installed, should generate a UUID [7] and then save this in persistent storage for all future use. For a device such as a hard phone, which will only ever have a single SIP UA present, the UUID can be generated at any time because it is guaranteed that no other UUID is being generated at the same time on that physical device. This means the value of the time component of the UUID can be arbitrarily selected to be any time less than the time when the device was manufactured. A time of 0 (as shown in the example in Section 3.2) is perfectly legal as long as the device knows no other UUIDs were generated at this time.

4.2 Detecting Flow Failure

The UA needs to detect if a given flow has failed, and if it does fail, follow the procedures in Section 4.1 to form a new flow to replace the failed one.

User Agents that form flows with stream oriented protocols such as TCP, TLS, or SCTP SHOULD periodically send a CRLF over the connection to detect liveness of the flow. If when sending the CRLF, the transport reports an error, then the connection is considered to have failed. It is RECOMMENDED that a CRLF be sent if the flow has not had any data sent or received in the previous 500 to 600 seconds. The exact time in the 500 to 600 second range SHOULD be randomly selected. These times MAY be configurable.

User Agents that form flows with datagram oriented protocols such as UDP SHOULD check if the URI has the "stun" tag (defined in Section 10) and, if the tag is present, then the UA needs to periodically perform STUN [5] requests over the flow. The time between STUN request SHOULD be a random number between 25 and 30 seconds. The times MAY be configurable. If the mapped address in

the STUN response changes, the UA must treat this as a failure on the flow.

Any time a SIP message is sent and the proxy does not respond, this is also considered a failure, the flow is closed and the procedures in Section 4.1 are followed to form a new flow.

4.3 Flow Failure Recovery

When a flow to a particular URI in the proxy set fails, the UA needs to form a new flow to replace it. The new flow MUST have the same flow-id as the flow it is replacing. This is done in much the same way as the forming flows described in Section 4.1; however, if there is a failure in forming this flow, the UA needs to wait a certain amount of time before retrying to form a flow to this particular URI in the proxy set. The time to wait is computed in the following way. If all of the flows to every URI in the proxy set have failed, the base time is set to 30 seconds; otherwise, in the case where at least one of the flows has not failed, the base time is set to 90 seconds. The wait time is computed by taking the minimum of 1800 seconds, or the base time multiplied by two to power of the number of consecutive registration failures to that URI.

```
wait-time = min( 1800, (30 * (2 ^ consecutive-failures)))
```

These three times SHOULD be configurable in the UA. For example if the base time was 30 seconds, and there had been three failures, then the wait time would be min(1800,30*(2^3)) or 240 seconds. The delay time is computed by selecting a uniform random time between 50 and 100 percent of the the wait time. The UA MUST wait for the value of the delay time before trying another registration to form a new flow for that URI.

To be explicitly clear on the boundary conditions, when the UA boots it immediately tries to register. If this fails and no registration on other flows had succeeded, the first retry would happen somewhere between 30 and 60 seconds after the failure of the first registration request.

4.4 Registration by other other instances

A User Agent MUST NOT include an instance-id or flow-id in the Contact header field of a registration if the registering UA is not the same instance as the UA referred to by the target Contact. (This practice is occasionally used to install forwarding policy into registrars.)

5. Registrar Mechanisms

5.1 Processing Register Requests

Registrars which implement this specification, processes REGISTER requests as described in Section 10 of RFC 3261 with the following change. Any time the registrar checks if a new contact matches an existing contact in the location database, it MUST also check and see if both the instance-id and flow-id match. If they do not match, then the they are not the same contact. The registrar MUST be prepared to receive some registrations that use instance-id and flow-id and some that do not, simultaneously for the same AOR.

In addition to the normal information stored in the binding record, some additional information MUST be stored for any registration that contains a flow-id header parameter in the Contact header field value. The registrar MUST store enough information to uniquely identify the network flow over which the request arrived. For common operating systems with TCP, this would typically just be the file descriptor. For common operating systems with UDP this would typically be the file descriptor for the local socket that received the request and the IP address and port number of the remote side that sent the request.

The registrar MUST also store all the Contact header field information including the flow-id and instance-id and SHOULD also store the time at which the binding was last updated. If the registrar receives a re-registration, it MUST update the information that uniquely identifies the network flow over which the request arrived and the time the binding was last updated.

5.2 Forwarding Requests

When a proxy uses the location service to look up a registration binding and then proxies a request to a particular contact, it selects a contact to use normally, with a few additional rules:

- The proxy MUST NOT populate the target set with more than one contact with the same AOR and instance-id at a time. If a request for a particular AOR and instance-id fails with a 410 response, the proxy SHOULD replace the failed branch with another target with the same AOR and instance-id, but a different flow-id.
- If two bindings have the same instance-id and flow-id, it MUST prefer the contact that was most recently updated.

Note that if the request URI is a GRUU, the proxy will only select contacts with the AOR and instance-id associated with the GRUU. The rules above still apply to a GRUU. This allows a request routed to a

GRUU to first try one of the flows to a UA, then if that fails, try another flow to the same UA instance.

Proxies MUST Record-Route so that mid dialog requests are routed over the correct flow.

When the proxy forwards a request to a binding that contains a flow-id, the proxy MUST send the request over the same network flow that was saved with the binding. For TCP, the request MUST be sent on the same TCP socket that received the REGISTER request. For UDP, the request MUST be sent from the same local IP address and port over which the registration was received to the same IP address and port from which the REGISTER was received.

If a proxy or registrar receives a network error when sending a SIP message over a particular flow, it MUST remove all the bindings that use that flow (regardless of AOR). Similarly, if a proxy closes a file descriptor, it MUST remove all the bindings that use that flow.

6. Edge Proxy Mechanisms

6.1 Processing Register Requests

When an edge proxy receives a registration request it MUST form a flow identifier token that is unique to this network flow and use this token as the user part of the URI that this proxy inserts into the Path header. A trivial way to satisfy this requirement involves storing a mapping between an incrementing counter and the connection information, however this would require the edge proxy to keep an impractical amount of state. It is unclear when this state could be removed and the approach would have problems if the proxy crashed and lost the value of the counter. Two stateless examples are provided below. A proxy can use any algorithm it wants as long as the flow token is unique to a flow.

Algorithm 1: The proxy generates a flow token for connection-oriented transports by concatenating the file descriptor (or equivalent) with the NTP time the connection was created, and base64 encoding the result. This results in an approximately 16 octet identifier. The proxy generates a flow token for UDP by concatenating the file descriptor and the remote IP address and port, then base64 encoding the result.

Algorithm 2: When the proxy boots it selects a 20 byte crypto random key called K that only the edge proxy knows. A byte array, called S, is formed that contains the following information about the flow the request was received on: an enumeration indicating the protocol, the local IP address and port, the remote IP address and port. The HMAC of S is computed using the key K and the HMAC-

SHA1-80 algorithm, as defined in [9]. The concatenation of the HMAC and S are base64 encoded, as defined in [10], and used as the flow identifier. With IPv4 address, this will result in a 32 octet identifier.

6.2 Forwarding Requests

When the edge proxy receives a request that is routed to a URI with a flow identifier token that this proxy created, then the proxy MUST forward the request over the flow that received the REGISTER request that caused the flow identifier token to be created. For connection-oriented transports, if the flow no longer exists the proxy SHOULD send a 410 response to the request. The advantage to a stateless approach to managing the flow information is that there is no state on the edge proxy that requires clean up that has to be synchronized with the registrar.

Algorithm 1: The proxy base64 decodes the user part of the Route header. For TCP, if a connection specified by the file descriptor is present and the creation time of the file descriptor matches the creation time encoded in the Route header, then proxy forwards the request over that connection. For UDP, the proxy forwards the request from the encoded file descriptor to the source IP address and port.

Algorithm 2: To decode the flow token take the flow identifier in the user portion of the URI, and base64 decode it, then verity the HMAC is correct by recomputing the HMAC and checking it matches. If the HMAC is not correct, the proxy SHOULD send a 403 response. If the HMAC was correct then the proxy should forward the request on the flow that was specified by the information in the flow identifier. If this flow no longer exists, the proxy SHOULD send a 410 response to the request.

Edge Proxies MUST Record-Route with the same URI that was used in the path so that mid dialog requests still are routed over the correct flow.

7. Mechanisms for All Servers

A SIP device that receives UDP datagrams directly from a UA needs to behave as specified in this section. Such devices would generally include a Registrar and an Edge Proxy, as they both receive register requests directly from a UA.

If the server receives UDP SIP requests on a given interface and port, it MUST also provide a limited version of the STUN server on the same interface and port. Specifically it MUST be capable of receiving and responding to UDP STUN requests with the exception that

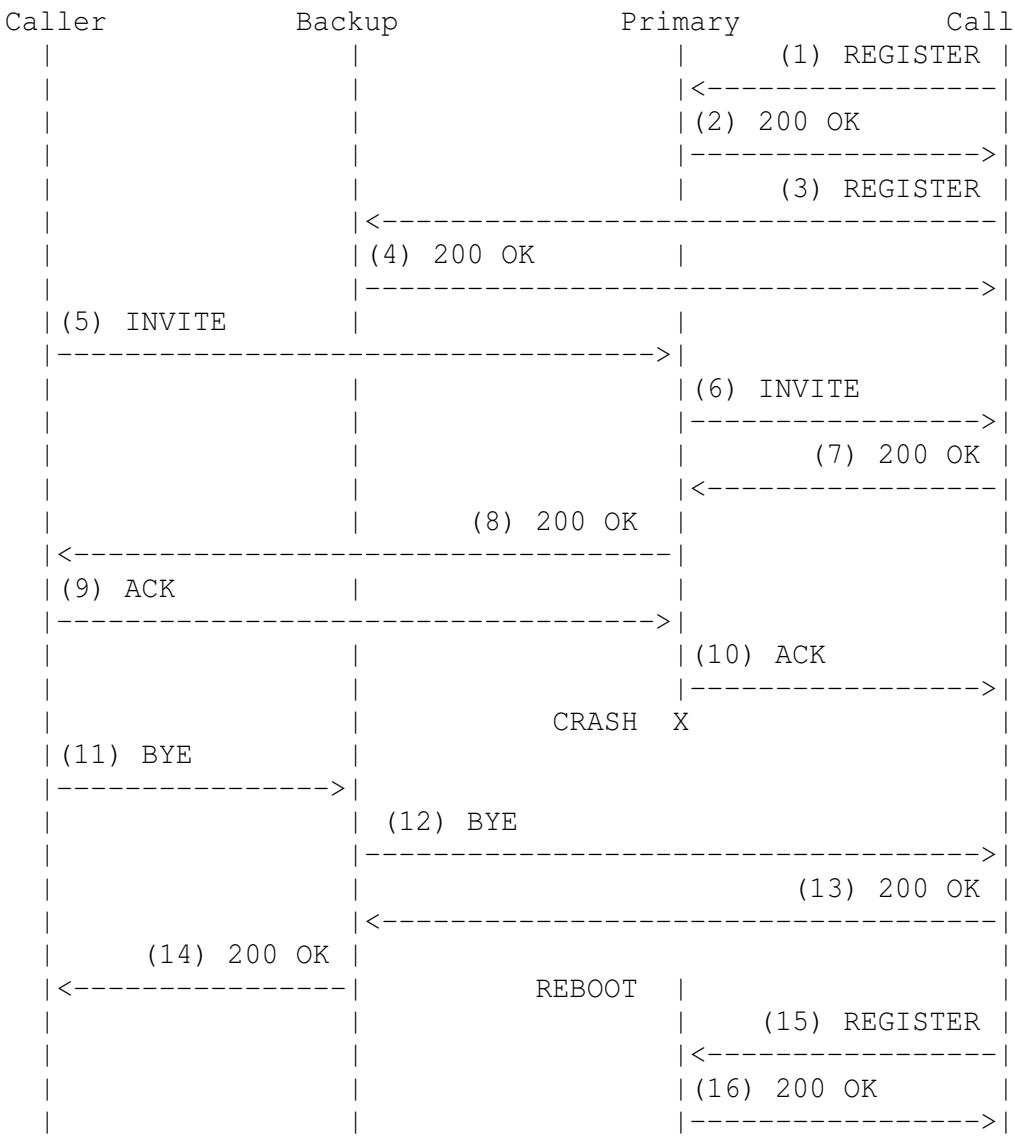
it does not need to support STUN requests with the changed port or changed address flag set. This allows the STUN server to run with only one port and IP address.

It is easy to distinguish STUN and SIP packets because the first octet of a STUN packet has a value of 0 or 1 while the first octet of a SIP message never a 0 or 1.

When a URI is created that refers to a SIP device that supports STUN as described in this section, the URI parameter "stun", as defined in Section 10 SHOULD be added to the URI. This allows a UA to inspect the URI to decide if it should attempt to send STUN requests to this location.

8. Example Message Flow

The following call flow shows a basic registration and an incoming call. Part way through the call, the flow to the Primary proxy is lost. The BYE message for the call is rerouted to the callee via the Backup proxy. When connectivity to the primary proxy is established, the Callee registers again to replace the lost flow as shown in message 15.



This call flow assumes that the Callee has been configured with a proxy set of that consists of "sip:primary.example.com;lr;stun" and "sip:backup.example.com;lr;stun". The Callee REGISTER in message (1) looks like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 10.0.1.1;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com>
Call-ID: 1j9FpLxk3uxtm8tn@10.0.1.1
CSeq: 1 REGISTER
Route: <sip:primary.example.com;lr>
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-id=1
Content-Length: 0
```

In the message, note that the Route is set and the Contact header field value contains the instance-id and flow-id. The response to the REGISTER in message (2) would look like:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.0.1.1;branch=z9hG4bKnashds7
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com> ;tag=b88sn
Call-ID: 1j9FpLxk3uxtm8tn@10.0.1.1
CSeq: 1 REGISTER
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-id=1
  ;expires=3600
Content-Length: 0
```

The second registration in message 3 and 4 are similar other than the Call-ID has changed, the flow-id is 2, and the route is set to the backup instead of the primary. They look like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 10.0.1.1;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com>
Call-ID: 1j9FpLxk3uxtm8tn-2@10.0.1.1
CSeq: 1 REGISTER
Route: <sip:primary.example.com;lr>
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-id=2
```

```
Content-Length: 0



SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.0.1.1;branch=z9hG4bKnashds7
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com> ;tag=b88sn
Call-ID: 1j9FpLxk3uxtm8tn-2@10.0.1.1
CSeq: 1 REGISTER
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-id=1
  ;expires=3600
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-id=2
  ;expires=3600
Content-Length: 0
```

The messages in the call flow are very normal. The only interesting thing to note is that the INVITE has a:

```
Record-Route: <sip:example.com;lr>
```

The registrations in message 15 and 16 are the same as message 1 and 2 other than the Call-ID has changed.

9. Grammar

This specification defines a new Contact header field parameter, flow-id. The grammar for DIGIT and EQUAL is obtained from RFC 3261 [3].

```
 contact-params = c-p-q / c-p-expires / c-p-flow / contact-extension
 c-p-flow       = "flow-id" EQUAL 1*DIGIT
```

The value of the flow-id MUST NOT be 0 and MUST be less than 2**31.

10. IANA Considerations

This specification defines a new Contact header field parameter called flow-id in the "Header Field Parameters and Parameter Values" sub-registry as per the registry created by [12] at http://www.iana.org/assignments/sip-parameters. The required information is:

Header Field	Parameter Name	Predefined Values	Reference
Contact	flow-id	Yes	[RFC AAAA]

[NOTE TO IANA: Please replace AAAA with the RFC number of this specification.]

This specification defines a new value in the "SIP/SIPS URI Parameters" sub-registry as per the registry created by [13] at http://www.iana.org/assignments/sip-parameters. The required information is:

Parameter Name	Predefined Values	Reference
stun	No	[RFC AAAA]

[NOTE TO IANA: Please replace AAAA with the RFC number of this specification.]

11. Security Considerations

One of the key security concerns in this work is making sure that an attacker cannot hijack the sessions of a valid user and cause all calls destined to that user to be sent to the attacker.

The simple case is when there are no edge proxies. In this case, the only time an entry can be added to the routing for a given AOR is when the registration succeeds. SIP protects against attackers being able to successfully register, and this scheme relies on that security. Some implementers have considered the idea of just saving the instance-id without relating it to the AOR with which it registered. This idea will not work because an attacker's UA can impersonate a valid user's instance-id and hijack that user's calls.

The more complex case involves one or more edge proxies. The only time an edge proxy will route over a particular flow is when it has received a route header that has the instance-id information it has created. An incoming request would have gotten this information from the registrar. The registrar will only save this information for a given AOR if the registration for the AOR has been successful; and the registration will only be successful if the UA can correctly authenticate. Even if an attacker has spoofed some bad information in the path header sent to the registrar, the attacker will not be able to get the registrar to accept this information for an AOR that does not belong to the attacker. The registrar will not hand out

this bad information to others, and others would not be misled into contacting the attacker.

12. Open Issues

This specification requires Record Routing to force flows through proxies. If all UA were required to implement GRUU, and all deployments were mandated to use GRUU, and there could never be a proxy behind a NAT or Firewall or deployed without a TLS certificate, then it would not be necessary to require the Record Routing. Should we do this?

The two algorithm for edge proxies are nearly identical with the exception that one integrity protects the identifier so it can not be tampered with. It is not clear if this integrity protection is needed. The WG should determine if this integrity is need or not then refine this specification.

13. Requirements

This specification was developed to meet the following requirements:

1. Must be able to detect that a UA supports these mechanisms.
2. Support UAs behind NATs.
3. Support TLS to a UA without a stable DNS name or IP.
4. Detect failure of connection and be able to correct for this.
5. Support many UAs simultaneously rebooting.
6. Support a NAT rebooting or resetting.
7. Support proxy farms with multiple hosts for scaling and reliability purposes.
8. Minimize initial startup load on a proxy.
9. Support proxies that provide geographic redundancy.
10. Support architectures with edge proxies.

14. Changes from 01 Version

Changed the algorithm and timing for retries of re-registrations.

Changed to using sigcomp style URI parameter to detect it - UA should attempt STUN to proxy.

Changed to use a configured set of backup proxies instead of playing DNS games to try and figure out what backup proxies to use.

15. Changes from 00 Version

Changed the behavior of the proxy so that it does not automatically remove registrations with the same instance-id and flow-id but

instead just uses the most recently created registration first.

Changed the connection-id to flow-id.

Fixed expiry of edge proxies and rewrote mechanism section to be clearer.

16. Acknowledgments

Jonathan Rosenberg provided many comments and useful text. Dave Oran came up with the idea of using the most recent registration first in the proxy. Alan Hawrylyshen helped with text on drafts that led to this one. Additionally, many of the concepts here originated at a connection reuse meeting at IETF 60 that included the authors, Jon Peterson, Jonathan Rosenberg, Alan Hawrylyshen, and Paul Kyzivat. The TCP design team consisting of Chris Boulton, Scott Lawrence, Rajnish Jain, Vijay K. Gurbani, and Ganesh Jayadevan provided input. In addition, thanks to the following folks for useful comments: Francois Audet, Flemming Andreasen, Dan Wing, Srivatsa Srinivasan, and Lyndsay Campbell.

17. References

17.1 Normative References

[1] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-04 (work in progress), July 2005.

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

[3] 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.

[4] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[5] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003.

[6] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[7] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[8] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

17.2 Informative References

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

[10] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003.

[11] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[12] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.

[13] Camarillo, G., "The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP)", BCP 99, RFC 3969, December 2004.

[14] Mahy, R., "Connection Reuse in the Session Initiation Protocol (SIP)", draft-ietf-sip-connect-reuse-03 (work in progress), October 2004.

[15] Mahy, R., "Requirements for Connection Reuse in the Session Initiation Protocol (SIP)", draft-ietf-sipping-connect-reuse-reqs-00 (work in progress), October 2002.

Authors' Addresses

Cullen Jennings (editor)
Cisco Systems
170 West Tasman Drive
Mailstop SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 902-3341
Email: fluffy@cisco.com

Rohan Mahy (editor)
SIP Edge LLC
5617 Scotts Valley Drive, Suite 200
Scotts Valley, CA 95066
USA

Email: rohan@ekabal.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP Working Group
Internet-Draft
Updates: 3515 (if approved)
Expires: January 6, 2006

O. Levin
Microsoft Corporation
A. Johnston
MCI

July 5, 2005

Conveying Feature Tags with Session Initiation Protocol REFER Method
draft-ietf-sip-refer-feature-param-00

Status of this Memo

Copyright Notice

Abstract

This document extends the SIP REFER method, defined in RFC 3515, to convey feature parameters defined in RFC 3840.

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 [1].

To simplify discussions of the REFER method and its extensions, three new terms are being used throughout the document:
- REFER-Issuer: the UA issuing the REFER request
- REFER-Recipient: the UA receiving the REFER request
- REFER-Target: the UA designated in the Refer-To URI

2. Introduction

This document extends the SIP [2] REFER method defined in RFC 3515 [3] to be used with feature parameters defined in RFC 3840 [4].

Feature tags are used by a SIP User Agent (UA) to convey to another UA information about capabilities and features. This information can be shared by a UA using a number of mechanisms including registration requests, OPTIONS responses, or shared in the context of a dialog by inclusion with a remote target URI (Uniform Resource Identifier), such as a Contact URI.

Feature tag information can be very useful to another UA. It is especially useful prior to the establishment of a session. For example, if a UA knows (through an OPTIONS query, for example) that the remote UA supports both video and audio, the calling UA might call offering video in its session description. Another example is when a UA knows that a remote UA is acting as a focus and hosting a conference. In this case, the UA might first subscribe to the conference URI and find out details about the conference prior to sending an INVITE to join.

This extension to the REFER method provides a mechanism by which the REFER-Issuer can provide this useful information about the REFER-Target capabilities and functionality to the REFER-Recipient by including feature tags in the Refer-To header field in a REFER request.

3. Definitions

The Refer-To BNF from RFC 3515:

```
Refer-To = ("Refer-To" / "r") HCOLON ( name-addr / addr-spec )
           *(SEMI generic-param)
```

is extended to:

```
Refer-To = ("Refer-To" / "r") HCOLON ( name-addr / addr-spec )
           *(SEMI refer-param)
refer-param = generic-param / feature-param
```

where feature-param is defined in Section 9 of RFC 3840 [4].

Note that if any URI parameters are present, the entire URI must be enclosed in "<" and ">". If no "<" and ">" are present, all parameters after the URI are header parameters, not URI parameters.

4. Examples

4.1 isfocus Feature Tag Usage

The example below shows how the "isfocus" feature tag can be used by REFER-Issuer to tell the REFER-Recipient that the REFER-Target is a conference focus and, consequently, sending an INVITE will bring the REFER-Recipient into the conference:

```
Refer-To: <sip:conf44@example.com>;isfocus
```

4.2 Voice and Video Feature Tags Usage

The example below shows how a REFER-Issuer can tell the REFER-Recipient that the REFER-Target supports audio and video and, consequently, that a video and audio session can be established by sending an INVITE to the REFER-Target:

```
Refer-To: "Alice's Videophone" <sip:alice@vphone.example.com>
 ;audio;video
```

4.3 Example with URI parameters and multiple feature tags

The example below shows how the REFER-Issuer can tell the REFER-Recipient that the REFER-Target is a voicemail server. Note that the transport URI parameter is enclosed within the "<" and ">" so that it is not interpreted as a header parameter.

```
Refer-To: <sip:alice-vm@example.com;transport=tcp>
 ;actor="msg-taker";automata;audio
```

5. IANA Considerations

This document requires no actions by IANA. Note that this document does not define any elements in the SIP Header Parameter Registry [5], since it incorporates media feature parameters instead of SIP header parameters.

6. Security Considerations

Feature tags can provide sensitive information about a user or a UA. As such, RFC 3840 cautions against providing sensitive information to another party. Once this information is given out, any use may be made of it, including relaying to a third party as in this specification.

As a result, it is NOT RECOMMENDED that all feature tag information be passed using the mechanism described in this specification. Instead, only feature tags that directly relate to a requested operation should be used. For example, the "isfocus" feature tag has clear operation semantics and utility. However, the "mobility" or "class" feature tags have no obvious use in a REFER scenario and should not be included unless their application is defined in the future.

A feature tag provided by a REFER-Issuer can not be authenticated or certified directly from the REFER request. As such, the REFER-Recipient MUST treat the information as hint. If the REFER-Recipient application logic or user's action depends on the presence of the expressed feature, the feature tag can be verified. For example, in order to do so, the REFER-Recipient can directly send an OPTIONS query to the REFER-Target over a secure (e.g. mutually authenticated and integrity protected) connection. This protects the REFER-Recipient against incorrect or malicious feature tags being sent.

A REFER-Issuer MUST NOT create or guess feature tags - instead a feature tag included in a REFER SHOULD have been discovered in an authenticated and secure method (such as an OPTIONS response or from a remote target URI in a dialog) directly from the REFER-Target.

7. Acknowledgements

The authors would like to thank Jonathan Rosenberg for providing helpful guidance to this work.

8. References

8.1 Normative References

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

[2] 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.

[3] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[4] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

8.2 Informative References

[5] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.

Authors' Addresses

Orit Levin
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

Phone: 425-722-2225
Email: oritl@microsoft.com

Alan Johnston
MCI
100 South 4th Street
St. Louis, MO 63102

Email: alan.johnston@mci.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 18, 2006

O. Levin
Microsoft Corporation

July 17, 2005

Suppression of Session Initiation Protocol REFER Method Implicit Subscription
draft-ietf-sip-refer-with-norefersub-02

Status of this Memo

Copyright Notice

Abstract

This specification defines a way to suppress an implicit subscription with the Session Initiation Protocol (SIP) REFER method. A new SIP option tag "norefersub" is defined to indicate support for this extension. A new SIP header field "Refer-Sub" is defined to request the usage of this extension.

Table of Contents

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 [1].

To simplify discussions of the REFER method and its extensions, the three terms below are being used throughout the document:

- REFER-Issuer: the UA issuing the REFER request
- REFER-Recipient: the UA receiving the REFER request
- REFER-Target: the UA designated in the Refer-To URI

2. Introduction

The REFER specification specifies that every REFER creates an implicit subscription between the REFER-Issuer and the REFER-Recipient.

This document defines a new SIP header field: "Refer-Sub" meaningful within a REFER transaction only. This header field, when set to "false", specifies that a REFER-Issuer requests that the REFER-Recipient doesn't establish an explicit subscription and the resultant dialog.

This document defines a new option tag: "norefersub". This tag, when included in the Supported header field, indicates that a User Agent (UA) is capable of accepting a REFER request without creating an implicit subscription when acting as a REFER-Recipient.

3. Motivation

The REFER specification mandates that every REFER creates an implicit subscription between the REFER-Issuer and the REFER-Recipient. This subscription results in at least one NOTIFY being sent from the REFER-Recipient to the REFER-Issuer. The REFER-Recipient may choose to cancel the implicit subscription with this NOTIFY. The REFER-Issuer may choose to cancel this implicit subscription with an explicit SUBSCRIBE (Expires: 0) after receipt of the initial NOTIFY.

One purpose of requiring the implicit subscription and initial NOTIFY is to allow for the situation where the REFER request gets forked and the REFER-Issuer needs a way to see the multiple dialogs that may be established as a result of the forked REFER. This is the same approach used to handle forking of SUBSCRIBE [4] requests. Where the REFER-Issuer explicitly specifies that forking not occur, the requirement that an implicit subscription be established is unnecessary.

Another purpose of the NOTIFY is to inform the REFER-Issuer of the progress of the SIP transaction that results from the REFER at the REFER-Recipient. In the case where the REFER-Issuer is already aware of the progress of the requested operation, such as when the REFER-Issuer has an explicit subscription to the dialog event package at the REFER-Recipient, the implicit subscription and resultant NOTIFY traffic related to the REFER can create an unnecessary network overhead.

4. Definitions

This document defines a new SIP header field: "Refer-Sub". This header field is meaningful and MAY be used with a REFER request and the corresponding 2XX response only. This header field set to "false" specifies that a REFER-Issuer requests that the REFER-Recipient doesn't establish an explicit subscription and the resultant dialog. Note that when using this extension, the REFER remains a target refresh request (as in the default case - when the extension is not used).

This document adds the following entry to Table 2 of [2]. The additions to this table are also provided for extension methods at the time of publication of this document. This is provided as a courtesy to the reader and is not normative in any way:

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	MSG
Refer-Sub	R, 2xx		-	-	-	-	-	-	-

Header field	where	SUB	NOT	REF	INF	UPD	PRA	PUB
Refer-Sub	R, 2xx	-	-	o	-	-	-	-

The Refer-Sub header field MAY be encrypted as part of end-to-end encryption.

The syntax of the header field follows the BNF defined below:

```
Refer-Sub            =  "Refer-Sub" HCOLON refer-sub-value extension-value
refer-sub-value      =  "true" / "false"
extension-value      =  *(TEXT-UTF8char / UTF8-CONT / LWS)
```

The "Refer-Sub" header field set to "false" MAY be used by the REFER-Issuer only when the REFER-Issuer can be certain that the REFER request will not be forked.

If the REFER-Recipient supports the extension and is willing to

process the REFER transaction without establishing an implicit subscription, it MUST insert the "Refer-Sub" header field set to "false" in the 2xx response to the REFER-Issuer. In this case no implicit subscription is created. Consequently, no new dialog is created if this REFER was issued outside any existing dialog.

If the REFER-Issuer inserts the "Refer-Sub" header field set to "false", but the REFER-Recipient doesn't grant the suggestion (i.e. either does not include the "Refer-Sub" header field or includes the "Refer-Sub" header field set to "true" in the 2xx response), an implicit subscription is created as in default case.

This document also defines a new option tag, "norefersub". This tag, when included in the Supported header field, specifies that a User Agent (UA) is capable of accepting a REFER request without creating an implicit subscription when acting as a REFER-Recipient.

If the capabilities of the REFER-Recipient are not known, using the "norefersub" tag with the Require header field is NOT RECOMMENDED. This is due to the fact that in the event the REFER-Recipient doesn't support the extension, in order to fallback to the normal REFER, the REFER-Issuer will need to issue a new REFER transaction thus resulting in additional round-trips.

The "norefersub" tag, when included in the Require header field (always in conjunction with the "Refer-Sub" header field set to "false"), specifies that the REFER-Recipient MUST process a REFER transaction without establishing an explicit subscription. In this case, if the REFER-Recipient either doesn't support the extension or is not willing to grant the request, the REFER request MUST be rejected by sending "420 Bad Extension" response back to the REFER-Issuer.

5. Preventing Forking of REFER Requests

The REFER specification allows for the possibility of forking a REFER request which is sent outside of an existing dialog. In addition, a proxy may fork an unknown method type. Should forking occur, the sender of the REFER with "Refer-Sub" will not be aware as only a single 2xx response will be forwarded by the forking proxy. As a result, the responsibility is on the issuer of the REFER with "Refer-Sub" to ensure that no forking will result.

The best way that the REFER-Issuer can ensure that REFER doesn't get forked is by only sending a REFER with "Refer-Sub" with a Request-URI which has GRUU properties according to definitions of [5].

If this is not known, the other way to ensure that forking will not

occur is to ensure that there are no proxies between the REFER-Issuer and the REFER-Recipient. This could be done by sending the REFER with a Max-Forwards: 0 header field. Any proxy receiving this request will return a "483 Too Many Hops" response, indicating that it is not safe to use this extension.

6. Example

An example of REFER which suppresses the implicit subscription is shown below:

```
REFER sip:pc-b@example.com SIP/2.0
Via: SIP/2.0/TCP issuer.example.com;branch=z9hG4bK-a-1
From: <sip:a@example.com>;tag=1a
To: <sip:pc-b@example.com>
Call-ID: 1@issuer.example.com
CSeq: 234234 REFER
Max-Forwards: 70
Refer-To: <sip:c@example.com;method=INVITE>
Refer-Sub: false
Supported: norefersub
Contact: sip:a@issuer.example.com
Content-Length: 0
```

7. IANA Considerations

This document registers a new SIP header field "Refer-Sub". This header field is only meaningful for the REFER request defined in RFC 3515 [3] and the corresponding response. The following information to be added to the header field sub-registry under http://www.iana.org/assignments/sip-parameters:

- Header Name: Refer-Sub
- Compact Form: None
- Reference: [Substitute with this RFC number]

This document also registers a new SIP option tag, "norefersub". The required information for this registration, as specified in RFC 3261 [2], is:

- Name: norefersub
- Description: This option tag specifies a User Agent ability of accepting a REFER request without establishing an implicit subscription (compared to the default case defined in RFC 3515 [3]).

8. Security Considerations

The purpose of this SIP extension is to modify the expected behavior

of the REFER-Recipient. The change in behavior is for the REFER-Recipient to not establish a dialog and to not send NOTIFY messages back to the REFER-Issuer. As such, a malicious inclusion of a "Refer-Sub" header field set to "false" reduces the processing and state requirements on the recipient. As a result, its use in a denial of service attack seems limited.

Should an intermediary maliciously insert a "Refer-Sub" header field set to "false", two possibilities may occur. If the REFER-Recipient does not support the extension, the REFER will fail with a "420 Bad Extension" response. The REFER-Issuer will be confused as no "Refer-Sub" was in the request, and the resulting request will fail. Should the REFER-Recipient support the extension, the 2xx response will contain the "Refer-Sub" header field set to "false". In any case, the REFER-Recipient will not establish a new dialog and send NOTIFYs. As a result the REFER-Recipient will not learn the outcome of the operation on the Refer-To URI.

Should an intermediary maliciously remove a "Refer-Sub" header field set to "false", the REFER-Recipient will try to sent notifications over the "explicitly established" dialog. It may confuse the REFER-Issuer, unless the Man in the Middle (MitM) has the motivation and the ability to intercept the notifications.

To protect against these kinds of MitM attacks, integrity protection should be used. For example, the REFER-Issuer could use S/MIME as discussed in RFC 3261 [2] to protect against these kinds of attacks.

9. Acknowledgements

The SIP community would like to thank Sriram Parameswar for his ideas being originally presented in draft-parameswar-sipping-norefersub-00 and served as the basis for this specification.

10. References

10.1 Normative References

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

[2] 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.

[3] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[4] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

10.2 Informational References

[5] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-04 (work in progress), July 2005.

Author's Address

Orit Levin
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

Phone: 425-722-2225
Email: oritl@microsoft.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 19, 2006

J. Rosenberg
Cisco Systems

July 18, 2005

Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)
draft-ietf-sip-target-dialog-01

Status of this Memo

Copyright Notice

Abstract

This specification defines the Target-Dialog header field for the Session Initiation Protocol (SIP), and the corresponding option tag, tdialog. This header field is used in requests that create SIP dialogs. It indicates to the recipient that the sender is aware of an existing dialog with the recipient, either because the sender is on the other side of that dialog, or because it has access to the dialog identifiers. The recipient can then authorize the request based on this awareness.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] defines the concept of a dialog as a persistent relationship between a pair of user agents. Dialogs provide context, including sequence numbers, proxy routes, and dialog identifiers. Dialogs are established through the transmission of SIP requests with particular methods. Specifically, the INVITE, REFER [7], SUBSCRIBE and NOTIFY [2] requests all create dialogs.

When a user agent receives a request that creates a dialog, it needs to decide whether to authorize that request. For some requests, authorization is a function of the identity of the sender, the request method, and so on. However, many situations have been identified in which a user agents' authorization decision depends on whether the sender of the request is currently in a dialog with that user agent, or whether the sender of the request is aware of a dialog the user agent has with another entity.

One such example is call transfer, accomplished through REFER. If user agents A and B are in an INVITE dialog, and user agent A wishes to transfer user agent B to user agent C, user agent A needs to send a REFER request to user agent B, asking user agent B to send an INVITE request to user agent C. User agent B needs to authorize this REFER. The proper authorization decision is that user agent B should accept the request if it came from a user with whom B currently has an INVITE dialog relationship. Current implementations deal with this by sending the REFER on the same dialog as the one in place between user agents A and B. However, this approach has numerous problems [9]. These problems include difficulty in determining the lifecycle of the dialog and its usages, and difficulties in determining which messages are associated with each application usage. Instead, a better approach is for user agent A to send the REFER request to user agent C outside of the dialog using its Globally Routable User Agent URI (GRUU) [10]. In that case, a means is needed for user agent B to authorize the REFER.

Another example is the application interaction framework [11]. In that framework, proxy servers on the path of a SIP INVITE request can place user interface components on the user agent that generated or received the request. To do this, the proxy server needs to send a REFER request to the user agent, targeted to their GRUU, asking the user agent to fetch an HTTP resource containing the user interface component. In such a case, a means is needed for the user agent to authorize the REFER. The appplication interaction framework recommends that the request be authorized if it was sent from an entity on the path of the original dialog. This can be done by including the dialog identifiers in the REFER, which prove that the

user agent that sent the REFER is aware of those dialog identifiers (this needs to be secured against eavesdroppers through the sips mechanism, of course)

Another example is if two user agents share an INVITE dialog, and an element on the path of the INVITE request wishes to track the state of the INVITE. In such a case, it sends a SUBSCRIBE request to the GRUU of the user agent, asking for a subscription to the dialog event package. If the SUBSCRIBE request came from an element on the INVITE request path, it should be authorized.

2. Overview of Operation

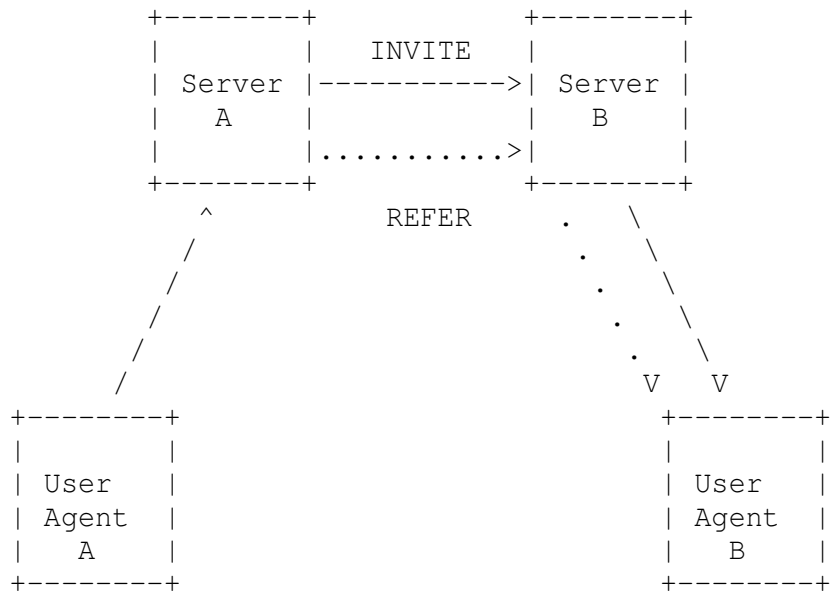


Figure 1

Figure 1 shows the basic model of operation. User agent A sends an INVITE to user agent B, traversing two servers, server A and server B. Both servers act as proxies for this transaction. User B sends a 200 OK response to the INVITE. This 200 OK includes a Supported header field indicating support for both the GRUU specification (through the presence of the gruu option tag) and this specification (through the presence of the tdialog option tag). The 200 OK

response establishes a dialog between the two user agents. Next, server A wishes to REFER user agent B to fetch an HTTP resource. So, it acts as a user agent and sends a REFER request to user agent B. This REFER is addressed to the GRUU of user agent B, which server A learned from inspecting the Contact header field in the 200 OK of the INVITE request. This GRUU is a URI that can be used by any element on the Internet, such as server A, to reach the specific user agent instance that generated that 200 OK to the INVITE.

The REFER request generated by server A will contain a Target-Dialog header field. This header field contains the dialog identifiers for the INVITE dialog between user agents A and B, composed of the Call-ID, local tag, and remote tag. Server A knew to include the Target-Dialog header field in the REFER request because it knows that user agent B supports it.

When the REFER request arrives at user agent B, it needs to make an authorization decision. Because the INVITE dialog was established using a sips URI, and because the dialog identifiers are cryptographically random [1], no entity except for user agent A or the proxies on the path of the initial INVITE request can know the dialog identifiers. Thus, because the REFER request contains those dialog identifiers, user agent B can be certain that the REFER request came from either user agent A, the two proxies, or an entity to whom the user agent or proxies gave the dialog identifiers. As such, it authorizes the REFER request, and fetches the HTTP resource identified by the URI of the Refer-To header field in the REFER request.

3. UAC Behavior

A UAC SHOULD include a Target-Dialog header field in a request if the following conditions are all true:

1. The request is to be sent outside of any existing dialog.

2. The user agent client believes that the request will not be authorized by the user agent server unless the user agent client can prove that it is aware of the dialog identifiers for some other dialog. Call this dialog the target dialog.

3. The request does not otherwise contain information that indicates that the UAC is aware of those dialog identifiers.

4. The user agent client knows that the user agent server supports the Target-Dialog header field. It can know this if it has seen a request or response from the user agent server within the target dialog that contained a Supported header field which

included the tdialog option tag.

If the fourth condition is not met, the UAC SHOULD NOT use this specification. Instead, if it is currently within a dialog with the UAS, it SHOULD attempt to send the request within the existing target dialog.

The following are examples of use cases in which these conditions are met:

- A REFER request is sent according to the principles of [11]. These REFER are sent outside of a dialog, and do not contain any other information which indicates awareness of the target dialog. [11] also mandates that the REFER be sent only if the UA indicates support for the target dialog specification.

- User A is in separate calls with users B and user C. It decides to start a three way call, and so morphs into a focus [14]. User B would like to learn the other participants in the conference. So, it sends a SUBSCRIBE request to user A (who is now acting as the focus) for the conference event package [13]. It is sent outside of the existing dialog between user B and the focus, and would be authorized by A if user B could prove that it knows the dialog identifiers for its existing dialog with the focus. Thus, the Target-Dialog header field would be include in the SUBSCRIBE.

The following are examples of use cases in which these conditions are not met:

- A server acting as a proxy is a participant in an INVITE dialog that establishes a session. The server would like to use the Keypad Markup Language (KPML) event package [15] to find out about keypresses from the originating user agent. To do this, it sends a SUBSCRIBE request. However, the Event header field of this SUBSCRIBE contains event parameters which indicate the target dialog of the subscription. As such, the request can be authorized without additional information.

- A server acting as a proxy is a participant in an INVITE dialog that establishes a session. The server would like to use the dialog event package [12] to find out about keypresses from the originating user agent. To do this, it sends a SUBSCRIBE request. However, the Event header field of this SUBSCRIBE contains event parameters which indicate the target dialog of the subscription. As such, the request can be authorized without additional information.

Specifications which intend to make use of the Target-Dialog header

field SHOULD discuss speific conditions in which it is to be included.

Assuming it is to be included, the value of the call-id production in the Target-Dialog header field MUST be equal to the Call-ID of the target dialog. The "remote-tag" header field parameter MUST be present, and MUST contain the tag that would be viewed as the remote tag from the perspective of the recipient of the new request. The "local-tag" header field parameter MUST be present, and MUST contain the tag that would be viewed as the local tag from the perspective of the recipient of the new request.

The request sent by the UAC SHOULD include a Require header field that includes the tdialog option tag. This request should, in principle, never fail with a 420 (Bad Extension) response, because the UAC would not have sent the request unless it believed the UAS supported the extension. If a Require header field was not included, and the UAS didn't support the extension, it would normally reject the request becaust it was unauthorized, probably with a 403. However, without the Require header field, the UAC would not be able to differentiate a 403 that arrived because the UAS didn't actually understand the Target-Dialog header field (in which case the client should send the request within the target dialog if it can), from a 403 that arrived because the UAS understood the Target-Dialog header field, but elected not to authorize the request despite the fact that the UAC proved its awareness of the target dialog (in which case the client should not resend the request within the target dialog, even if it could).

4. User Agent Server Behavior

If a user agent server receives a dialog-creating request, and wishes to authorize the request, and that authorization depends on whether or not the sender has knowledge of an existing dialog with the UAS, and information outside of the Target-Dialog header field does not provide proof of this knowledge, the UAS SHOULD check the request for the existence of the Target-Dialog header field. If this header field is not present, the UAS MAY still authorize the request based on other means.

If the header field is present, and the value of the call-id production, the "remote-tag" and "local-tag" values match the Call-ID, remote tag and local tag of an existing dialog, and the dialog that they match was established using a sips URI, the UAS SHOULD authorize the request if it would authorize any entity on the path of the request that created that dialog, or any entity trusted by an entity on the path of the request that created that dialog.

If the dialog identifiers match, but they match a dialog not created with a sips URI, the UAS MAY authorize the request if it would authorize any entity on the path of the request that created that dialog, or any entity trusted by an entity on the path of the request that created that dialog. However, in this case, any eavesdropper on the original dialog path would have access to the dialog identifiers, and thus the authorization strength is reduced to MAY.

If the dialog identifiers don't match, or if they don't contain both a "remote-tag" and "local-tag" parameter, the header field MUST be ignored, and authorization MAY be determined by other means.

5. Proxy Behavior

Proxy behavior is unaffected by this specification.

6. Extensibility Considerations

This specification depends on a user agent client knowing, ahead of sending a request to a user agent server, whether or not that user agent server supports the Target-Dialog header field. As discussed in Section 3, the UAC can know this because it saw a request or response sent by that UAS within the target dialog that contained the Supported header field whose value included the tdialog option tag.

Because of this requirement, it is especially important that user agents compliant to this specification include a Supported header field in all dialog forming requests and responses. Inclusion of the Supported header fields in requests is at SHOULD strength within RFC 3261. This specification does not alter that requirement. However, implementors should realize that, unless the tdialog option tag is placed in the Supported header field of requests and responses, this extension is not likely to be used, and instead, the request is likely to be resent within the existing target dialog (assuming the sender is the UA on the other side of the target dialog). As such, the conditions in which the SHOULD would not be followed would be those rare cases in which the UA does not want to enable usage of this extension.

7. Header Field Definition

The grammar for the Target-Dialog header field is defined as follows:

```
Target-Dialog      =     "Target-Dialog" HCOLON call-id *(SEMI
                         td-param)
td-param           =     remote-param / local-param / generic-param
remote-param       =     "remote-tag" EQUAL token
```

```
local-param        =     "local-tag" EQUAL token
```

Figure 3 and Figure 4 are an extension of Tables 2 and 3 in RFC 3261 [1] for the Target-Dialog header field. The column "INF" is for the INFO method [3], "PRA" is for the PRACK method [4], "UPD" is for the UPDATE method [5], "SUB" is for the SUBSCRIBE method [2], "NOT" is for the NOTIFY method [2], "MSG" is for the MESSAGE method [6], "REF" is for the REFER method [7], and "PUB" is for the PUBLISH method [8].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	PUB
Target-Dialog	R	ar	-	-	-	o	-	-	-

Figure 3: Allowed Methods for Target-Dialog

Header field	where	proxy	PRA	UPD	SUB	NOT	INF	MSG	REF
Target-Dialog	R	ar	-	-	o	-	-	-	o

Figure 4: Allowed Methods for Target-Dialog

8. Security Considerations

The Target-Dialog header field is used to authorize requests based on the fact that the sender of the request has access to information that only certain entities have access to. In order for such an authorization decision to be secure, two conditions have to be met. Firstly, no eavesdroppers can have access to this information. That requires the original SIP dialog to be established using a sips URI, which provides TLS on each hop. With a sips URI, only the user agents and proxies on the request path will be able to know the dialog identifiers. The second condition is that the dialog identifiers be sufficiently random that they cannot be guessed. RFC 3261 requires global uniquess for the Call-ID and 32 bits of randomness for each tag (there are two tags for a dialog). Given the short duration over which a typical dialog exists (perhaps as long as a day), this amount of randomness appears adequate to prevent guessing attacks.

9. Example Call Flow

In this example, user agent A and user agent B establish an INVITE initiated dialog through Server-A and Server-B, each of which acts as a proxy for the INVITE. Server B would then like to use the app interaction framework [11] to request user agent A to fetch an HTML

user interface component. To do that, it sends a REFER request to A's GRUU. The flow for this is shown in Figure 5. The conventions of [16] are used to describe representation of long message lines.

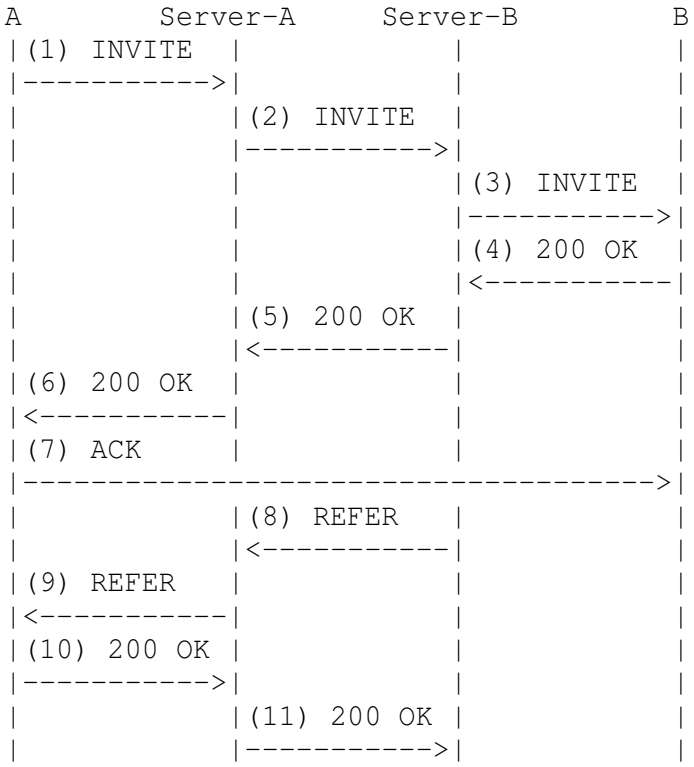


Figure 5

First, the caller sends an INVITE, as shown in message 1.

```
INVITE sips:B@example.com SIP/2.0
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.org>
Call-ID: fa77as7dad8-sd98ajzz@host.example.com
CSeq: 1 INVITE
Max-Forwards: 70
Supported: gruu, tdialog
Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, REFER
Accept: application/sdp, text/html
<allOneLine>
Contact: <sips:A@example.com;opaque=urn:uuid:f81d4f
ae-7dec-11d0-a765-00a0c91e6bf6;grid=99a>;schemes="http,sip,sips"
</allOneLine>
Content-Length: ...
Content-Type: application/sdp

--SDP not shown--
```

The INVITE indicates that the caller supports GRUU (note its presence in the Contact header field of the INVITE) and the Target-Dialog header field. This INVITE is forwarded to the callee (messages 2-3), which generates a 200 OK response that is forwarded back to the caller (message 4-5). Message 5 might look like:

```
SIP/2.0 200 OK
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.org>;tag=6544
Call-ID: fa77as7dad8-sd98ajzz@host.example.com
CSeq: 1 INVITE
Contact: <sips:B@pc.example.org>
Content-Length: ...
Content-Type: application/sdp

--SDP not shown--
```

In this case, the called party does not support GRUU or the Target-Dialog header field. The caller generates an ACK (message 7). Server B then decides to send a REFER to user A:

```
   <allOneLine>
   REFER sips:A@example.com;opaque=urn:uuid:f81d4f
   ae-7dec-11d0-a765-00a0c91e6bf6;grid=99a SIP/2.0
   </allOneLine>
   Via: SIP/2.0/TLS serverB.example.org;branch=z9hG4bK9zz10
   From: Server B <sip:serverB.example.org>;tag=mreysh
   <allOneLine>
   To: Caller <sips:A@example.com;opaque=urn:uuid:f81d4f
   ae-7dec-11d0-a765-00a0c91e6bf6;grid=99a>
   </allOneLine>
   Target-Dialog: fa77as7dad8-sd98ajzz@host.example.com
     ;local-tag=kkaz-
     ;remote-tag=6544
   Refer-To: http://serverB.example.org/ui-component.html
   Call-ID: 86d65asfklzll8f7asdr@host.example.com
   CSeq: 1 REFER
   Max-Forwards: 70
   Require: tdialog
   Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, NOTIFY
   Event: refer
   Contact: <sips:serverB.example.org>
   Content-Length: 0
```

This REFER will be delivered to server A because it was sent to the GRUU. From there, it is forwarded to user agent A (message 9), and authorized because of the presence of the Target-Dialog header field.

10. IANA Considerations

This specification registers a new SIP header field and a new option tag according to the processes of RFC 3261 [1].

10.1 Header Field

RFC Number: RFC XXXX [Note to IANA: Fill in with the RFC number of this specification.]

Header Field Name: Target-Dialog

Compact Form: none

10.2 SIP Option Tag

This specification registers a new SIP option tag per the guidelines in Section 27.1 of RFC 3261.

Name: tdialog

Description: This option tag is used to identify the target dialog header field extension. When used in a Require header field, it implies that the recipient needs to support the Target-Dialog header field. When used in a Supported header field, it implies that the sender of the message supports it.

11. Acknowledgments

This specification is based on a header field first proposed by Robert Sparks in the dialog usage draft. John Elwell provided helpful comments.

12. References

12.1 Normative References

[1] 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.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[4] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

[5] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

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

[7] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[8] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

12.2 Informative References

[9] Sparks, R., "Multiple Dialog Usages in the Session Initiation Protocol", draft-sparks-sipping-dialogusage-00 (work in progress), July 2004.

[10] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-03 (work in progress), February 2005.

[11] Rosenberg, J., "A Framework for Application Interaction in the Session Initiation Protocol (SIP)", draft-ietf-sipping-app-interaction-framework-04 (work in progress), February 2005.

[12] Rosenberg, J., "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-06 (work in progress), April 2005.

[13] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Conference State", draft-ietf-sipping-conference-package-12 (work in progress), July 2005.

[14] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-05 (work in progress), May 2005.

[15] Burger, E., "A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)", draft-ietf-sipping-kpml-07 (work in progress), December 2004.

[16] Sparks, R., "Session Initiation Protocol Torture Test Messages", draft-ietf-sipping-torture-tests-07 (work in progress), May 2005.

Author's Address

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 18, 2006

C. Jennings
Cisco Systems
J. Peterson
NeuStar, Inc.
July 17, 2005

Certificate Management Service for The Session Initiation Protocol (SIP)
draft-ietf-sipping-certs-02

Status of this Memo

Copyright Notice

Abstract

This draft defines a Credential Service that allows SIP User Agents to use a SIP package to discover the certificates of other users. This mechanism allows user agents that want to contact a given Address-of-Record (AOR) to retrieve that AOR's certificate by subscribing to the Credential Service. The Credential Service also allows users to store and retrieve their own certificates and private keys.

Table of Contents

1. Introduction

SIP [6] provides a mechanism [18] for end-to-end encryption and integrity using S/MIME [17]. Several security properties of SIP depend on S/MIME, and yet it has not been widely deployed. Certainly, one reason is the complexity of providing a reasonable certificate distribution infrastructure. This specification proposes a way to address discovery, retrieval, and management of certificates for SIP deployments. It follows the Sacred Framework RFC 3760 [7] for management of the credentials. Combined with the SIP Identity [2] specification, this specification allows users to have certificates that are not signed by any well known certificate authority while still strongly binding the user's identity to the certificate. This mechanism allows SIP User Agents such as IP phones to enroll and get their credentials without any more configuration information than they commonly have today. The end user expends no extra effort.

2. 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 RFC 2119 [5].

Certificate: An X.509v3 [15] style certificate containing a public key and a list of identities in the SubjectAltName that are bound to this key. The certificates discussed in this draft are generally self signed and use the mechanisms in the SIP Identity [2] specification to vouch for their validity, but certificates that are signed by a certificate authority can also be used with all the mechanisms in this draft.

Credential: For this document, credential means the combination of a certificate and the associated private key.

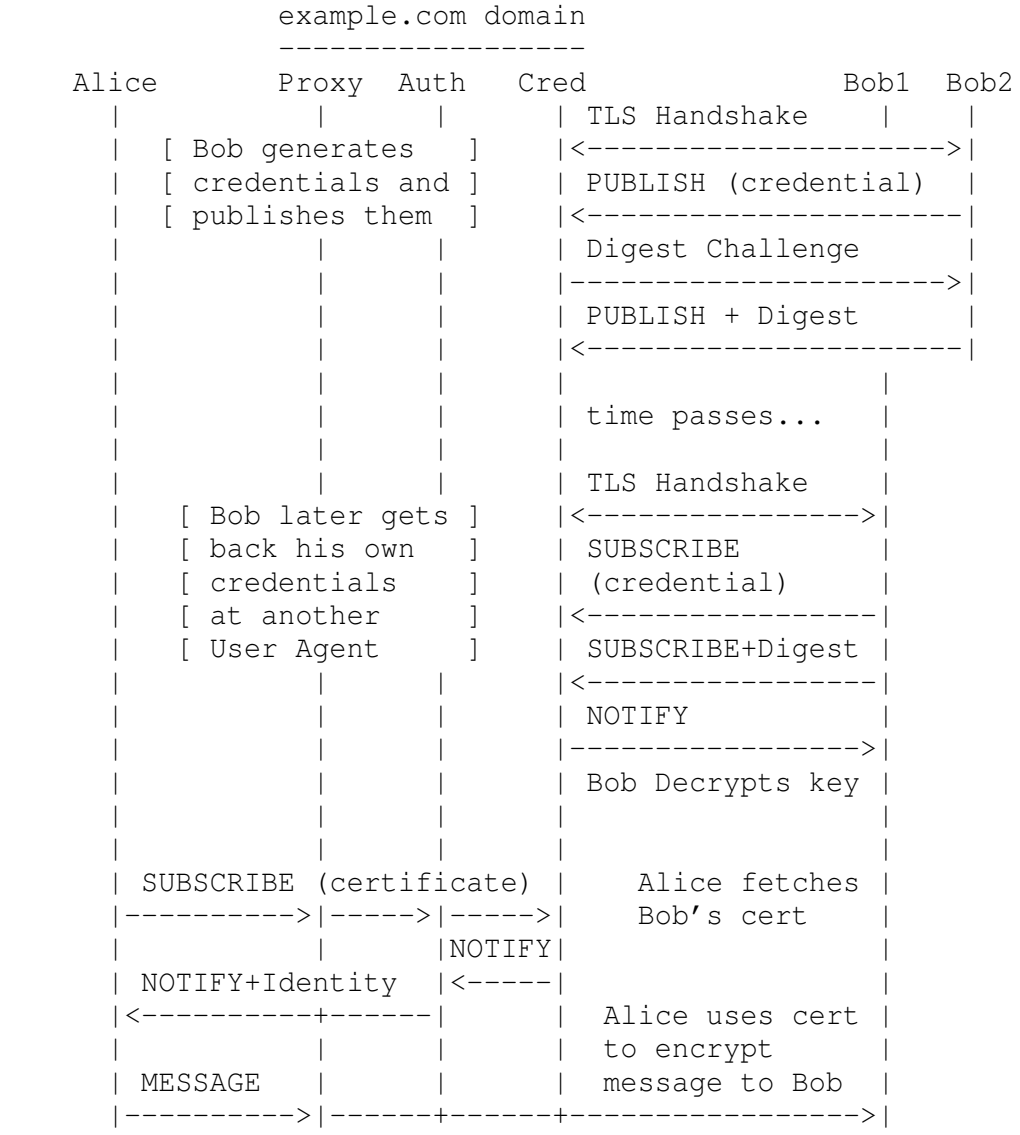
password phrase: A password used to encrypt a PKCS#8 private key.

3. Overview

The general approach is to provide a new SIP service referred to as a "credential service" that allows SIP User Agents (UAs) to subscribe to other users' certificates using a new SIP event package [4]. The certificate is delivered to the subscribing UA in a corresponding SIP NOTIFY request. The identity of the certificate can be vouched for using the Authentication Service from the SIP Identity [2] specification, which uses the domain's certificate to sign the NOTIFY request. The credential service can manage public certificates as well as the user's private keys. Users can update their credentials, as stored on the credential service, using a SIP PUBLISH [3] request. The UA authenticates to the credential service using a shared secret

when a UA is updating a credential. Typically the shared secret will be the same one that is used by the UA to authenticate a REGISTER request with the Registrar for the domain (usually with SIP Digest Authentication).

The following figure shows Bob publishing his credentials from one of his User Agents (e.g. his laptop software client), retrieving his credentials from another of his User Agents (e.g. his mobile phone), and then Alice retrieving Bob's certificate and sending a message to Bob. SIP 200-class responses are omitted from the diagram to make the figure easier to understand.



Bob's UA (Bob2) does a TLS [11] handshake with the credential server

to authenticate that the UA is connected to the correct credential server. Then Bob's UA publishes his newly created or updated credentials. The credential server digest challenges the UA to authenticate that the UA knows Bob's shared secret. Once the UA is authenticated, the credential server stores Bob's credentials.

Another of Bob's User Agents (Bob1) wants to fetch its current credentials. It does a TLS [11] handshake with the credential server to authenticate that the UA is connected to the correct credential server. Then Bob's UA subscribes for the credentials. The credential server digest challenges the UA to authenticate that the UA knows Bob's shared secret. Once the UA is authenticated, the credential server sends a NOTIFY that contains Bob's credentials. The private key portion of the credential may have been encrypted with a secret that only Bob's UA (and not the credential server) knows. In this case, once Bob's UA decrypts the private key it will be ready to go. Typically Bob's UA would do this when it first registered on the network.

Some time later Alice decides that she wishes to discover Bob's certificate so that she can send him an encrypted message or so that she can verify the signature on a message from Bob. Alice's UA sends a SUBSCRIBE message to Bob's AOR. The proxy in Bob's domain routes this to the credential server via an authorization service. The credential server returns a NOTIFY that contains Bob's public certificate in the body. This is routed through an authentication service that signs that this message really can validly claim to be from the AOR "sip:bob@example.com". Alice's UA receives the certificate and can use it to encrypt a message to Bob.

It is critical to understand that the only way that Alice can trust that the certificate really is the one for Bob and that the NOTIFY has not been spoofed is for Alice to check that the Identity [2] header field value is correct.

The mechanism described in this document works for both self signed certificates and certificates signed by well known certificate authorities; however, it is imagined that most UAs using this would only use self signed certificates and would use an Authentication Service as described in [2] to provide a strong binding of an AOR to the certificates.

The mechanisms described in this draft allow for three different styles of deployment:

1. Deployments where the the credential server only stores certificates and does not store any private key information. If the deployment had users with multiple devices, some other scheme

(perhaps even manual provisioning) would be used to get the right private keys onto all the devices that a user uses.

2. Deployments where the credential server stores certificates and also stores encrypted version of the private keys. The credential server would not know or need the password phrase for decrypting the private key. The credential server would help move the the private keys between devices but the user would need to enter a password phrase on each device to allow that device to decrypt (and encrypt) the private key information.
3. Deployments where the credential server stores the certificates and private keys and also knows the password phrase for decrypting the private keys. Deployments such as these may not even use password phrases, in which case the private keys are not encrypted inside the PKCS#8 objects. This style of deployments would often have the credential server, instead of the devices, create the credentials.

4. UA Behavior with Certificates

When a User Agent wishes to discover some other user's certificate it subscribes to the "certificate" SIP event package as described in Section 7 to get the certificate. While the subscription is active, if the certificate is updated, the Subscriber will receive the updated certificate in a notification.

The Subscriber needs to decide how long it is willing to trust that the certificate it receives is still valid. If the certificate is revoked before it expires, the Notifier will send a notification with an empty body to indicate that the certificate is no longer valid. However, the Subscriber might not receive the notification if an attacker blocks this traffic. The amount of time that the Subscriber caches a certificate SHOULD be configurable. A default of one day is RECOMMENDED.

Note that the actual duration of the subscription is orthogonal to the caching time or validity time of the corresponding certificate. Allowing subscriptions to persist after a certificate is not longer valid ensures that Subscribers receive the replacement certificate in a timely fashion. In some cases, the Notifier will not allow unauthenticated subscriptions to persist. The Notifier could return an immediate notification with the certificate in response to subscribe and then immediately terminate subscription, setting the reason parameter to "probation". The Subscriber will have to periodically poll the Notifier to verify validity of the certificate.

If the UA uses a cached certificate in a request and receives a 437 (Unsupported Certificate) response, it SHOULD remove the certificate it used from the cache, attempt to fetch the certificate again. If

the certificate is the not the same, then the UA SHOULD retry the original request again. This situation usually indicates that the certificate was recently updated, and that the Subscriber has not received a corresponding notification. If the certificate fetched is the same as the one that was previously in the the cache, then the UA SHOULD NOT try the request again. This situation can happened when the request was retargeted to a different user than the original request. The 437 response is defined in [2].

Note: A UA that has a presence list MAY want to subscribe to the certificates of all the presentities in the list when the UA subscribes to their presence, so that when the user wishes to contact a presentity, the UA will already have the appropriate certificate. Future specifications might consider the possibility of retrieving the certificates along with the presence documents.

The details of how a UA deals with receiving encrypted messages is outside the scope of this specification but it is worth noting that if Charlie's UAS receives a request that is encrypted to Bob, it would be valid and legal for that UA to send a 302 redirecting the call to Charlie.

5. UA Behavior with Credentials

UAs discover their own credentials by subscribing to their AOR with an event type of credential as described in Section 8. After a UA registers, it SHOULD retrieve its credentials by subscribing to them as described in Section 7.6.

When a UA discovers its credential, the private key information might be encrypted with a password phrase. The UA SHOULD request that the user enter the password phrase on the device, and the UA MAY cache this password phrase for future use.

There are several different cases in which a UA should generate a new credential:

- If the UA receives a NOTIFY with no body for the credential package.
- If the certificate has expired.
- If the certificate is within 600 seconds of expiring, the UA SHOULD attempt to create replacement credentials. The UA does this by waiting a random amount of time between 0 and 300 seconds. If no new credentials have been received in that time, the UA creates new credentials to replace the expiring ones and sends them in a PUBLISH request (with a SIP-If-Match header set to the current etag). This makes credential collisions both unlikely and harmless.

- o If the user of the device has indicated via the user interface that they wish to revoke the current certificate and issue a new one.

Credentials are created by creating a new key pair which will require appropriate randomness, and then creating a certificate as described in Section 10.6. The UA MAY encrypt the private key with a password phrase supplied by the user. Then the UA updates the user's credential by sending a PUBLISH [3] request with the credentials or just the certificate as described in Section 8.9.

If a UA wishes to revoke the existing certificate without publishing a new one, it MUST send a PUBLISH with an empty body to the credential server.

6. Credential Service Behavior

The credential service stores credentials for users and can provide the credentials to other user agents belonging to the same user, and certificates to any user agent. The credentials are indexed by a URI that corresponds to the AOR of the user. When a UA requests a public certificate with a SUBSCRIBE, the server sends the UA the certificate in a NOTIFY and sends a subsequent NOTIFY any time the certificate changes. When a credential is requested, the credential service digest challenges the requesting UA to authenticate it so that the credential service can verify that the UA is authorized to receive the requested credentials. When a credential is published, the credential service digest challenges the requesting UA to authenticate it so that the credential service can verify that the UA is authorized to change the credentials. This behavior is defined in Section 7 and Section 8.

7. Event Package Formal Definition for "certificate"

7.1 Event Package Name

This document defines a SIP Event Package as defined in RFC 3265 [4]. The event-package token name for this package is:

certificate

7.2 Event Package Parameters

This package does not define any event package parameters.

7.3 SUBSCRIBE Bodies

This package does not define any SUBSCRIBE bodies.

7.4 Subscription Duration

Subscriptions to this event package can range from no time to weeks. Subscriptions in days are more typical and are RECOMMENDED. The default subscription duration for this event package is one day.

The credential service is encouraged to keep the subscriptions active for AORs that are communicating frequently, but the credential service MAY terminate the subscription at any point in time.

7.5 NOTIFY Bodies

The body of a NOTIFY request for this package MUST either be empty or contain an application/pkix-cert body (as defined in [10]) that contains the certificate, unless an Accept header has negotiated some other type. The Content-Disposition MUST be set to "signal".

A future extension MAY define other NOTIFY bodies. If no "Accept" header is present in the SUBSCRIBE, the body type defined in this document MUST be assumed.

Implementations which generate large notifications are reminded to follow the message size restrictions for unreliable transports articulated in Section 18.1.1 of SIP.

7.6 Subscriber Generation of SUBSCRIBE Requests

A UA discovers a certificate by sending a SUBSCRIBE request with an event type of "certificate" to the AOR for which a certificate is desired. In general, the UA stays subscribed to the certificate for as long as it plans to use and cache the certificate, so that the UA can be notified about changes or revocations to the certificate.

Subscriber User Agents will typically subscribe to certificate information for a period of hours or days, and automatically attempt to re-subscribe just before the subscription is completely expired.

When a user de-registers from a device (logoff, power down of a mobile device, etc.), subscribers SHOULD unsubscribe by sending a SUBSCRIBE request with an Expires header of zero.

7.7 Notifier Processing of SUBSCRIBE Requests

When a SIP credential server receives a SUBSCRIBE request with the certificate event-type, it is not necessary to authenticate the subscription request. The Notifier MAY limit the duration of the subscription to an administrator-defined period of time. The duration of the subscription does not correspond in any way to the

period for which the certificate will be valid.

When the credential server receives a SUBSCRIBE request for a certificate, it first checks to see if it has credentials for the requested URI. If it does not have a certificate, it returns a NOTIFY request with an empty message body.

7.8 Notifier Generation of NOTIFY Requests

Immediately after a subscription is accepted, the Notifier MUST send a NOTIFY with the current certificate, or an empty body if no certificate is available for the target user. In either case it forms a NOTIFY with the From header field value set to the value of the To header field in the SUBSCRIBE request. This server sending the NOTIFY needs either to implement an Authentication Service (as described in SIP Identity [2]) or else the server needs to be set up such that the NOFIFY request will be sent through an Authentication Service. Sending the NOTIFY request through the the Authentication Service requires the SUBSCRIBE request to have been routed through the Authentication Service, since the NOTIFY is sent within the dialog formed by the subscription.

7.9 Subscriber Processing of NOTIFY Requests

The resulting NOTIFY will contain an application/pkix-cert body that contains the requested certificate. The UA MUST follow the procedures in Section 10.3 to decide if the received certificate can be used. The UA needs to cache this certificate for future use. The maximum length of time it should be cached for is discussed in Section 10.1. The certificate MUST be removed from the cache if the certificate has been revoked (if a NOTIFY with an empty body is received), or if it is updated by a subsequent NOTIFY. The UA MUST check that the NOTIFY is correctly signed by an Authentication Service as described in [2]. If the identity asserted by the Authentication Service does not match the AOR that the UA subscribed to, the certificate in the NOTIFY is discarded and MUST NOT be used.

7.10 Handling of Forked Requests

This event package does not permit forked requests. At most one subscription to this event type is permitted per resource.

7.11 Rate of Notifications

Notifiers SHOULD NOT generate NOTIFY requests more frequently than once per minute.

7.12 State Agents and Lists

Implementers MUST NOT implement state agents for this event type. Likewise, implementations MUST NOT use the event list extension [19] with this event type. It is not possible to make such an approach work, because the Authentication service would have to simultaneously assert several different identities.

7.13 Behavior of a Proxy Server

There are no additional requirements on a SIP Proxy, other than to transparently forward the SUBSCRIBE and NOTIFY requests as required in SIP. This specification describes the Proxy, Authentication service, and credential service as three separate services, but it is certainly possible to build a single SIP network element that performs all of these services at the same time.

8. Event Package Formal Definition for "credential"

8.1 Event Package Name

This document defines a SIP Event Package as defined in RFC 3265 [4]. The event-package token name for this package is:

credential

8.2 Event Package Parameters

This package defines the "etag" Event header parameter which is valid only in NOTIFY requests. It contains a token which represents the SIP etag value at the time the notification was sent. Considering how infrequently credentials are updated, this hint is very likely to be the correct etag to use in the SIP-If-Match header in a SIP PUBLISH request to update the current credentials.

```
etag-param = "etag" EQUAL token
```

8.3 SUBSCRIBE Bodies

This package does not define any SUBSCRIBE bodies.

8.4 Subscription Duration

Subscriptions to this event package can range from hours to one week. Subscriptions in days are more typical and are RECOMMENDED. The default subscription duration for this event package is one day.

The credential service SHOULD keep subscriptions active for UAs that are currently registered.

8.5 NOTIFY Bodies

The NOTIFY MUST contain a multipart/mixed (see [14]) body that contains both an application/pkix-cert body with the certificate and an application/pkcs8 body that has the associated private key information for the certificate. The Content-Disposition MUST be set to "signal" as defined in [16].

A future extension MAY define other NOTIFY bodies. If no "Accept" header is present in the SUBSCRIBE, the body type defined in this document MUST be assumed.

The application/pkix-cert body is a DER encoded X.509v3 certificate [10]. The application/pkcs8 body contains a DER-encoded PKCS#8 [1] object that contains the private key. The PKCS#8 objects MUST be of type PrivateKeyInfo. The integrity and confidentiality of the PKCS#8 objects is provided by the TLS transport. The transport encoding of all the MIME bodies is binary.

8.6 Subscriber Generation of SUBSCRIBE Requests

A Subscriber User Agent will subscribe to its credential information for a period of hours or days and will automatically attempt to re-subscribe before the subscription has completely expired.

The Subscriber SHOULD subscribe to its credentials whenever a new user becomes associated with the device (a new login). The subscriber SHOULD also renew its subscription immediately after a reboot, or when the subscriber's network connectivity has just been re-established.

The UA needs to authenticate with the credential service for these operations. The UA MUST use TLS to connect to the server. The UA may be configured with a specific name for the credential service; otherwise normal SIP routing is used. As described in RFC 3261, the TLS connection needs to present a certificate that matches the expected name of the server to which the connection was formed, so that the UA knows it is talking to the correct server. Failing to do this may result in the UA publishing its private key information to an attacker. The credential service will authenticate the UA using the usual SIP Digest mechanism, so the UA can expect to receive a SIP challenge to the SUBSCRIBE or PUBLISH requests.

8.7 Notifier Processing of SUBSCRIBE Requests

When a credential service receives a SUBSCRIBE for a credential, the credential service has to authenticate and authorize the UA and validate that adequate transport security is being used. Only a UA that can authenticate as being able to register as the AOR is authorized to receive the credentials for that AOR. The credential Service MUST digest challenge the UA to authenticate the UA and then decide if it is authorized to receive the credentials. If authentication is successful, the Notifier MAY limit the duration of the subscription to an administrator-defined period of time. The duration of the subscription MUST not be larger than the length of time for which the certificate is still valid. The Expires header should be set appropriately.

8.8 Notifier Generation of NOTIFY Requests

Once the UA has authenticated with the credential service and the subscription is accepted, the credential service MUST immediately send a Notify request. The Notifier SHOULD include the current etag value in the "etag" Event package parameter in the NOTIFY request. The Authentication Service is applied to this NOTIFY request in the same way as the certificate subscriptions. If the credential is revoked, the credential service MUST terminate any current subscriptions and force the UA to re-authenticate by sending a NOTIFY with its Subscription-State header set to "terminated" and a reason parameter of "deactivated". (This causes a Subscriber to retry the subscription immediately.) This is so that if a secret for retrieving the credentials gets compromised, the rogue UA will not continue to receive credentials after the compromised secret has been changed.

Any time the credentials for this URI change, the credential service MUST send a new NOTIFY to any active subscriptions with the new credentials.

8.9 Generation of PUBLISH Requests

A user agent SHOULD be configurable to control whether it publishes the credential for a user or just the user's certificate.

When publishing just a certificate, the body contains an application/pkix-cert. When publishing a credential, the body contains a multipart/mixed containing both an application/pkix-cert and an application/pkcs8 body.

When the UA sends the PUBLISH [3] request, it needs to do the

following:

- The Expires header field value in the PUBLISH request SHOULD be set to match the time for which the certificate is valid.
- If the certificate includes Basic Constraints, it SHOULD set the CA flag to false.
- The PUBLISH request SHOULD include a SIP-If-Match header field with the previous etag from the subscription. This prevents multiple User Agents for the same AOR from publishing conflicting credentials. Note that UAs replace credentials that are about to expire at a random time (described in Section 5), reducing the chance of publishing conflicting credentials even without using the etag.

8.10 Notifier Processing of PUBLISH Requests

When the credential service receives a PUBLISH to update credentials, it MUST authenticate and authorize this request the same way as for subscriptions for credentials. If the authorization succeeds, then the credential service MUST perform the following check on the the certificate:

- One of the names in the SubjectAltName of the certificate matches the authorized user making the request.
- The notBefore validity time MUST NOT be in the future.
- The notAfter validity time MUST be in the future.
- If an CA Basic Constraint is set in the certificate, it is set to false.

If all of these succeed, the credential service updates the credential for this URI, processes all the active certificates and credential subscriptions to this URI, and generates a NOTIFY request with the new credential or certificate.

If the Subscriber submits a PUBLISH request with no body, this revokes the current credentials and causes all subscriptions to the credential package to be deactivated as described in the previous section. (Note that subscriptions to the certificate package are NOT terminated; each subscriber to the certificate package receives a notification with an empty body.)

8.11 Subscriber Processing of NOTIFY Requests

When the UA receives a valid NOTIFY request, it should replace its existing credentials with the new received ones. If the UA cannot decrypt the PKCS#8 object, it MUST send a 437 (Unsupported Certificate) response. Later if the user provides a new password phrase for the private key, the UA can subscribe to the credentials again and attempt to decrypt with the new password phrase.

8.12 Handling of Forked Requests

This event package does not permit forked requests.

8.13 Rate of Notifications

Notifiers SHOULD NOT generate NOTIFY requests more frequently than once per minute.

8.14 State Agents and Lists

Implementers MUST NOT implement state agents for this event type. Likewise, implementations MUST NOT use the event list extension [19] with this event type.

8.15 Behavior of a Proxy Server

The behavior is identical to behavior described for certificate subscriptions described in Section 7.13.

9. Examples

In all these examples, large parts of the messages are omitted to highlight what is relevant to this draft. The lines in the examples that are prefixed by $ represent encrypted blocks of data.

9.1 Encrypted Page Mode IM Message

In this example, Alice sends Bob an encrypted page mode instant message. Alice does not already have Bob's public key from previous communications, so she fetches Bob's public key from Bob's credential service:

```
SUBSCRIBE sip:bob@biloxi.example.com SIP/2.0
...
Event: certificate
```

The credential service responds with the certificate in a NOTIFY.

```
NOTIFY alice@atlanta.example.com  SIP/2.0
Subscription-State: active; expires=7200
....
From: <sip:bob@biloxi.example.com>;tag=1234
Identity: "NJguAbpmYXjnlxFmlOkumMI+MZXjB2iV/NW5xsFQqzD/p4yiovrJBqhd3T
          ZkegnsmoHryzk9gTBH7Gj/erixEFIf82o3Anmb+CIbrgdl03gGaD6ICvkp
          VqoMXZZjdvSpycyHOhh1cmUx3b9Vr3pZuEh+cB01pbMQ8B1ch++iMjw="
Identity-Info: <https://atlanta.example.com/cert>;alg=rsa-sha1
....
Event: certificate
Content-Type: application/pkix-cert
Content-Disposition: signal

< certificate data >
```

Next, Alice sends a SIP MESSAGE message to Bob and can encrypt the body using Bob's public key as shown below. Although outside the scope of this document, it is worth noting that instant messages often have common plain text like "Hi", so that setting up symmetric keys for extended session mode IM conversations will likely increase efficiency, as well as reducing the likelihood of compromising the asymmetric key in the certificate.

```
MESSAGE sip:bob@biloxi.example.com SIP/2.0
...
Content-Type: application/pkcs7-mime
Content-Disposition: render

$ Content-Type: text/plain
$
$ < encrypted version of "Hello" >
```

9.2 Setting and Retrieving UA Credentials

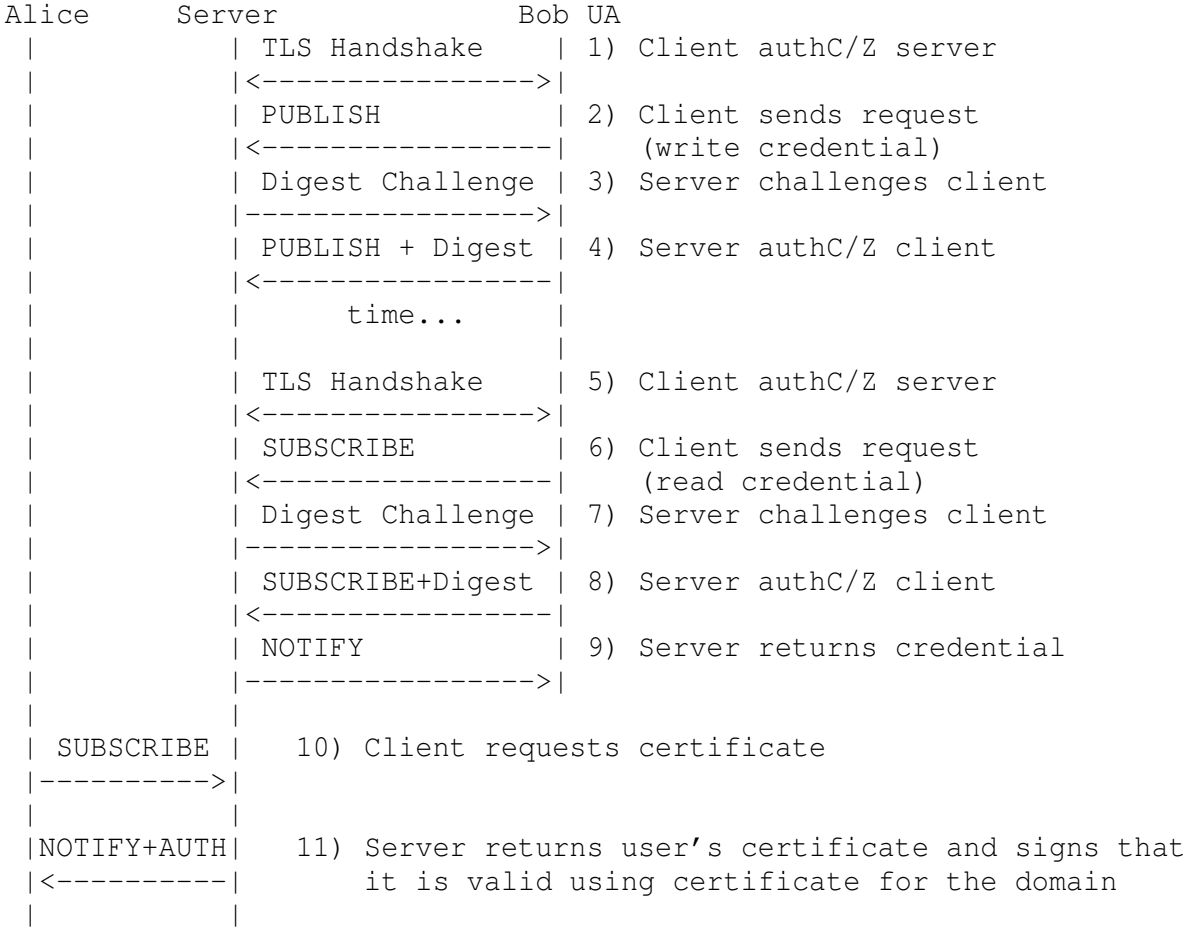
When Alice's UA wishes to publish Alice's public and private keys to the credential service, it sends a PUBLISH request like the one below. This must be sent over a TLS connection in which the other end of the connection presents a certificate that matches the credential service for Alice and digest challenges the request to authenticate her.

```
PUBLISH sips:alice@atlanta.example.com SIP/2.0
...
Content-Type: multipart/mixed;boundary=boundary
Content-Disposition: signal

--boundary
Content-ID: 123
Content-Type: application/pkix-cert


< Public certificate for Alice >
--boundary
Content-ID: 456
Content-Type: application/pkcs8

< Private Key for Alice >
--boundary
```

If one of Alice's UAs subscribes to the credential event, the UA will be digest challenged, and the NOTIFY will include a body similar to the one in the PUBLISH section above.

10. Security Considerations

The high level message flow from a security point of view is summarized in the following figure. The 200 responses are removed from the figure as they do not have much to do with the overall security.



When the UA, labeled Bob, first created a credential for Bob, it would store this on the the credential server. The UA authenticated the Server using the certificates from the TLS handshake. The Server authenticated the UA using a digest style challenge with a shared secret.

The UA, labeled Bob, wishes to request its credentials from the server. First it forms a TLS connection to the Server, which provides integrity and privacy protection and also authenticates the server to Bob's UA. Next the UA requests its credentials using a SUBSCRIBE request. The Server digest challenges this to authenticate Bob's UA. The server and Bob's UA have a shared secret that is used for this. If the authentication is successful, the server sends the credentials to Bob's UA. The private key in the credentials may have been encrypted using a shared secret that the server does not know.

A similar process would be used for Bob's UA to publish new credentials to the server. The SUBSCRIBE request would change to a PUBLISH request and there would not be an NOTIFY. When this

happened, all the other UAs that were subscribed to Bob's credentials would receive a new NOTIFY with the new credentials.

Alice wishes to find Bob's certificate and sends a SUBSCRIBE to the server. The server sends the response in NOTIFY. This does not need to be sent over a privacy or integrity protected channel, as the Authentication service described in [2] provides integrity protection of this information and signs it with the certificate for the domain.

This whole scheme is highly dependent on trusting the operators of the credential service and trusting that the credential service will not be compromised. The security of all the users will be compromised if the credential service is compromised.

> Note: There has been significant discussion of the topic of avoiding deployments in which the credential servers store the private keys, even in some encrypted form that the credential server does not know how to decrypt. Various schemes were considered to avoid this but they all result in either moving the problem to some other server, which does not seem to make the problem any better, or having a different credential for each device. For some deployments where each user has only one device this is fine but for deployments with multiple devices, it would require that when Alice went to contact Bob, Alice would have to provide messages encrypted for all of Bob's devices. The sipping working group did consider this architecture and decided it was not appropriate due both to the information it revealed about the devices and users and the amount of signaling required to make it work.

This specification requires the TLS session to be used for SIP communications to the credential service. As specified in RFC 3261, TLS clients MUST check that the SubjectAltName of the certificate for the server they connected to exactly matches the server they were trying to connect to. Failing to use TLS or selecting a poor cipher suite (such as NULL encryption) will result in credentials, including private keys, being sent unencrypted over the network and will render the whole system useless. Implementations really must use TLS or there is no point in implementing any of this.

The correct checking of chained certificates as specified in TLS [11] is critical for the client to authenticate the server. If the client does not authenticate that it is talking to the correct credential service, a man in the middle attack is possible.

10.1 Certificate Revocation

If a particular credential needs to be revoked, the new credential is

simply published to the credential service. Every device with a copy of the old credential or certificate in its cache will have a subscription and will rapidly (order of seconds) be notified and replace its cache. Clients that are not subscribed will subscribe when they next need to use the certificate and will get the new certificate.

It is possible that an attacker could mount a DOS attack such that the UA that had cached a certificate did not receive the NOTIFY with its revocation. To protect against this attack, the UA needs to limit how long it caches certificates. After this time, the UA would invalidate the cached information even though no NOTIFY had ever been received due to the attacker blocking it.

The duration of this cached information is in some ways similar to a device deciding how often to check a CRL list. For many applications, a default time of 1 day is suggested, but for some applications it may be desirable to set the time to zero so that no certificates are cached at all and the credential is checked for validity every time the certificate is used.

10.2 Certificate Replacement

The UAs in the system replace the certificates close to the time that the certificates would expire. If a UA has used the same key pair to encrypt a very large volume of traffic, the UA MAY choose to replace the credential with a new one before the normal expiration.

10.3 Trusting the Identity of a Certificate

When a UA wishes to discover the certificate for sip:alice@example.com, the UA subscribes to the certificate for alice@example.com and receives a certificate in the body of a SIP NOTIFY request. The term original URI is used to describe the URI that was in the To header field value of the SUBSCRIBE request. So in this case the original URI would be sip:alice@example.com.

If the certificate is signed by a trusted CA, and one of the names in the SubjectAltName matches the original URI, then this certificate MAY be used but only for exactly the original URI and not for other identities found in the SubjectAltName. Otherwise, there are several steps the UA MUST perform before using this certificate.

- The From header in the NOTIFY request MUST match the original URI that was subscribed to.
- The UA MUST check the Identity header as described in the Identity [2] specification to validate that bodies have not been tampered with and that an Authentication Service has validated this From header.

- The UA MUST check the validity time of the certificate and stop using the certificate if it is invalid. (Implementations are reminded to verify both the notBefore and notAfter validity times.)
- The certificate MAY have several names in the SubjectAltName but the UA MUST only use this certificate when it needs the certificate for the identity asserted by the Authentication Service in the NOTIFY. This means that the certificate should only be indexed in the certificate cache by the AOR that the Authentication Service asserted and not by the value of all the identities found in the SubjectAltName list.

These steps result in a chain of bindings that result in a trusted binding between the original AOR that was subscribed to and a public key. The original AOR is forced to match the From. The Authentication Service validates that this request did come from the identity claimed in the From header field value and that the bodies in the request that cary the certificate have not been tampered with. The certificate in the body contains the public key for the identity. Only the UA that can authenticate as this AOR, or devices with access to the private key of the domain, can tamper with this body. This stops other users from being able to provide a false public key. This chain of assertion from original URI, to From, to body, to public key is critical to the security of the mechanism described in this specification. If any of the steps above are not followed, this chain of security will be broken and the system will not work.

10.4 Conformity to the SACRED Framework

This specification uses the security design outlined in the SACRED Framework [7]. Specifically, it follows the cTLS architecture described in section 4.2.2 of RFC 3760. The client authenticates the server using the server's TLS certificate. The server authenticates the client using a SIP digest transaction inside the TLS session. The TLS sessions form a strong session key that is used to protect the credentials being exchanged.

10.5 Crypto Profiles

Credential services SHOULD implement the server name indication extensions in RFC 3546 [8] and they MUST support a TLS profile of TLS_RSA_WITH_AES_128_CBC_SHA as described in RFC 3268 [9] and a profile of TLS_RSA_WITH_3DES_EDE_CBC_SHA.

The PKCS#8 in the clients MUST implement PBES2 with a key derivation algorithm of PBKDF2 using HMAC with SHA1 and an encryption algorithm of DES-EDE2-CBC-Pad as defined in RFC 2898 [12]. It is RECOMMENDED that this profile be used when using PKCS#8.

10.6 User Certificate Generation

The certificates should be consistent with RFC 3280 [13]. A signatureAlgorithm of sha1WithRSAEncryption MUST be implemented. The Issuers SHOULD be the same as the subject. Given the ease of issuing new certificates with this system, the Validity can be relatively short. A Validity of one year or less is RECOMMENDED. The subjectAltName must have a URI type that is set to the SIP URL corresponding to the user AOR. It MAY be desirable to put some randomness into the length of time for which the certificates are valid so that it does not become necessary to renew all the certificates in the system at the same time.

It is worth noting that a UA can discover the current time by looking at the Date header field value in the 200 response to a REGISTER request.

10.7 Compromised Authentication Service

One of this worst attacks against this system would be if the Authentication Service were compromised. This attack is somewhat analogous to a CA being compromised in traditional PKI systems. The attacker could make a fake certificate for which it knows the private key, use it to receive any traffic for a given use, and then re-encrypt that traffic with the correct key and forward the communication to the intended receiver. The attacker would thus become a man in the middle in the communications.

There is not too much that can be done to protect against this. A UA MAY subscribe to its own certificate under some other identity to try to detect whether the credential server is handing out the correct certificates. It will be difficult to do this in a way that does not allow the credential server to recognize the user's UA.

The UA MAY also save the fingerprints of the cached certificates and warn users when the certificates change significantly before their expiry date.

The UA MAY also allow the user to see the fingerprints for the cached certificates so that they can be verified by some other out of band means.

11. IANA Considerations

This specification defines two new event packages that IANA is requested to add the registry at:

http://www.iana.org/assignments/sip-events
It also defines a new mime type that IANA is requested to add to the registry at:
http://www.iana.org/assignments/media-types/application

11.1 Certificate Event Package

To: ietf-sip-events@iana.org
Subject: Registration of new SIP event package

Package Name: certificate

Is this registration for a Template Package: No

Published Specification(s): This document

New Event header parameters: This package defines no new parameters

Person & email address to contact for further information:
Cullen Jennings <fluffy@cisco.com>

11.2 Credential Event Package

To: ietf-sip-events@iana.org
Subject: Registration of new SIP event package

Package Name: credential

Is this registration for a Template Package: No

Published Specification(s): This document

New Event header parameters: "etag"

Person & email address to contact for further information:
Cullen Jennings <fluffy@cisco.com>

11.3 PKCS#8

To: ietf-types@iana.org
Subject: Registration of MIME media type application/pkcs8

MIME media type name: application

MIME subtype name: pkcs8

Required parameters: None

Optional parameters: None

Encoding considerations: The PKCS#8 object inside this MIME type MUST be DER-encoded.

This MIME type was designed for use with protocols which can carry binary-encoded data. Protocols which do not carry binary data (which have line length or character-set restrictions for example) MUST use a reversible transfer encoding (such as base64) to carry this MIME type. Protocols that carry binary data SHOULD use a transfer encoding of "binary".

Security considerations: Carries a cryptographic private key

Interoperability considerations: None

Published specification:
RSA Laboratories, "Private-Key Information Syntax Standard, Version 1.2", PKCS 8, November 1993.

Applications which use this media type: Any MIME-compliant transport

Additional information:
Magic number(s): None
File extension(s): .p8
Macintosh File Type Code(s): none

Person & email address to contact for further information:
Cullen Jennings <fluffy@cisco.com>

Intended usage: COMMON

Author/Change controller:
the IESG

12. Acknowledgments

Many thanks to Eric Rescorla, Jim Schaad, Rohan Mahy for significant help and discussion. Many others provided useful comments, including Kumiko Ono, Peter Gutmann, Russ Housley, Yaron Pdut, Aki Niemi, Magnus Nystrom, Paul Hoffman, Adina Simu, Dan Wing, Mike Hammer, Lyndsay Campbell, and Jason Fischl. Rohan Mahy, John Elwell, and Jonathan Rosenberg provided detailed review and text.

13. References

13.1 Normative References

[1] RSA Laboratories, "Private-Key Information Syntax Standard, Version 1.2", PKCS 8, November 1993.

[2] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[3] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

[4] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

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

[6] 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.

[7] Gustafson, D., Just, M., and M. Nystrom, "Securely Available Credentials (SACRED) - Credential Server Framework", RFC 3760, April 2004.

[8] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

[9] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

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

[11] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[12] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[13] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[14] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[15] International Telecommunications Union, "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, ISO Standard 9594-8, March 2000.

[16] Zimmerer, E., Peterson, J., Vemuri, A., Ong, L., Audet, F., Watson, M., and M. Zonoun, "MIME media types for ISUP and QSIG Objects", RFC 3204, December 2001.

13.2 Informational References

[17] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

[18] Peterson, J., "S/MIME Advanced Encryption Standard (AES) Requirement for the Session Initiation Protocol (SIP)", RFC 3853, July 2004.

[19] Roach, A., Rosenberg, J., and B. Campbell, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", draft-ietf-simple-event-list-07 (work in progress), January 2005.

Authors' Addresses

Cullen Jennings
Cisco Systems
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

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

Jon Peterson
NeuStar, Inc.
1800 Sutter St
Suite 570
Concord, CA 94520
US

Phone: +1 925/363-8720
Email: jon.peterson@neustar.biz
URI: http://www.neustar.biz/

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP WG
Internet-Draft
Expires: January 17, 2006

C. Jennings
Cisco Systems
K. Ono
NTT Corporation

July 16, 2005

Example call flows using SIP security mechanisms
draft-jennings-sip-sec-flows-03

Status of this Memo

Copyright Notice

Abstract

This document shows call flows demonstrating the use of SIPS, TLS, and S/MIME in SIP. This draft provides information that helps implementers build interoperable SIP software. It is purely informational. To help facilitate interoperability testing, it includes certificates used in the example call flows and a CA certificate to create certificates for testing.

This work is being discussed on the sip@ietf.org mailing list.

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 RFC-2119 [1].

Table of Contents

1. Introduction

Several different groups are starting to implement the S/MIME[7] portion of SIP[2]. Over the last several interoperability events, it has become clear that it is difficult to write these systems without any test vectors or examples of "known good" messages to test against. Furthermore, testing at the events is often hampered by trying to get certificates signed by some common test root into the appropriate format for various clients. This document addresses both of these issues by providing detailed messages that give detailed examples that implementers can use for comparison and that can also be used for testing. In addition, this document provides a common certificate that can be used for a CA to reduce the time it takes to set up a test at an interoperability event. The document also provides some hints and clarifications for implementers.

A simple SIP call flow using SIPS and TLS is shown in Section 4. The certificates for the hosts used are shown in Section 3.2 and the CA certificates used to sign these are shown in Section 3.1.

The text from Section 5.1 through Section 5.3 shows some simple SIP call flows using S/MIME to sign and encrypt the body of the message. The user certificates used in these examples are shown in Section 3.3. These host certificates are signed with the same CA certificate.

Section 6 presents a partial list of things implementers should check that they do in order to implement a secure system.

A way to make certificates that can be used for interoperability testing is presented in Appendix A, along with methods for converting these to various formats.

The S/MIME messages shown in this document were made using client implementations from the authors' respective companies. These implementations are different code bases and though there may still be errors in these flows, the authors feel that the interoperability of these two clients bodes well for the correctness of the flows in this document.

2. Security Considerations

Implementers must never use any of the certificates provided in this document in anything but a test environment. Installing the CA root certificates used in this document as a trusted root in operational software would completely destroy the security of the system while giving the user the impression that the system was operating securely.

This document recommends some things that implementers might test or verify to improve the security of their implementations. It is impossible to make a comprehensive list of these, and this document only suggests some of the most common mistakes that have been seen at the SIPit interoperability events. Just because an implementation does everything this document recommends does not make it secure.

The S/MIME examples use 3DES, but AES is preferred.

3. Certificates

3.1 CA Certificates

The certificate used by the CA to sign the other certificates is shown below. This is a X509v3 certificate. Note that the basic constraints allow it to be used as a CA.

```
   Version: 3 (0x2)
   Serial Number: 0 (0x0)
   Signature Algorithm: sha1WithRSAEncryption
   Issuer: C=US, ST=California, L=San Jose, O=sipit,
           OU=Sipit Test Certificate Authority
   Validity
       Not Before: Jul 18 12:21:52 2003 GMT
       Not After : Jul 15 12:21:52 2013 GMT
   Subject: C=US, ST=California, L=San Jose, O=sipit,
           OU=Sipit Test Certificate Authority
   Subject Public Key Info:
       Public Key Algorithm: rsaEncryption
       RSA Public Key: (1024 bit)
           Modulus (1024 bit):
               00:c3:22:1e:83:91:c5:03:2c:3c:8a:f4:11:14:c6:
               4b:9d:fa:72:78:c6:b0:95:18:a7:e0:8c:79:ba:5d:
               a4:ae:1e:21:2d:9d:f1:0b:1c:cf:bd:5b:29:b3:90:
               13:73:66:92:6e:df:4c:b3:b3:1c:1f:2a:82:0a:ba:
               07:4d:52:b0:f8:37:7b:e2:0a:27:30:70:dd:f9:2e:
               03:ff:2a:76:cd:df:87:1a:bd:71:eb:e1:99:6a:c4:
               7f:8e:74:a0:77:85:04:e9:41:ad:fc:03:b6:17:75:
               aa:33:ea:0a:16:d9:fb:79:32:2e:f8:cf:4d:c6:34:
               a3:ff:1b:d0:68:28:e1:9d:e5
           Exponent: 65537 (0x10001)
   X509v3 extensions:
     X509v3 Subject Key Identifier:
       6B:46:17:14:EA:94:76:25:80:54:6E:13:54:DA:A1:E3:54:14:A1:B6
     X509v3 Authority Key Identifier:
       6B:46:17:14:EA:94:76:25:80:54:6E:13:54:DA:A1:E3:54:14:A1:B6
       DirName:/C=US/ST=California/L=San Jose/O=sipit/
                OU=Sipit Test Certificate Authority
       serial:00
     X509v3 Basic Constraints:
       CA:TRUE
   Signature Algorithm: sha1WithRSAEncryption
    96:6d:1b:ef:d5:91:93:45:7c:5b:1f:cf:c4:aa:47:52:0b:34:
    a8:50:fa:ec:fa:b4:2a:47:4c:5d:41:a7:3d:c0:d6:3f:9e:56:
    5b:91:1d:ce:a8:07:b3:1b:a4:9f:9a:49:6f:7f:e0:ce:83:94:
    71:42:af:fe:63:a2:34:dc:b4:5e:a5:ce:ca:79:50:e9:6a:99:
    4c:14:69:e9:7c:ab:22:6c:44:cc:8a:9c:33:6b:23:50:42:05:
    1f:e1:c2:81:88:5f:ba:e5:47:bb:85:9b:83:25:ad:84:32:ff:
    2a:5b:8b:70:12:11:83:61:c9:69:15:4f:58:a3:3c:92:d4:e8:
    6f:52
```

The ASN.1 parse of the CA certificate is shown below.

```
  0:l= 804 cons: SEQUENCE
  4:l= 653 cons:  SEQUENCE
  8:l=   3 cons:   cont [ 0 ]
 10:l=   1 prim:    INTEGER           :02
 13:l=   1 prim:   INTEGER           :00
 16:l=  13 cons:   SEQUENCE
 18:l=   9 prim:    OBJECT            :sha1WithRSAEncryption
 29:l=   0 prim:    NULL
 31:l= 112 cons:   SEQUENCE
 33:l=  11 cons:    SET
 35:l=   9 cons:     SEQUENCE
 37:l=   3 prim:      OBJECT            :countryName
 42:l=   2 prim:      PRINTABLESTRING   :US
 46:l=  19 cons:    SET
 48:l=  17 cons:     SEQUENCE
 50:l=   3 prim:      OBJECT            :stateOrProvinceName
 55:l=  10 prim:      PRINTABLESTRING   :California
 67:l=  17 cons:    SET
 69:l=  15 cons:     SEQUENCE
 71:l=   3 prim:      OBJECT            :localityName
 76:l=   8 prim:      PRINTABLESTRING   :San Jose
 86:l=  14 cons:    SET
 88:l=  12 cons:     SEQUENCE
 90:l=   3 prim:      OBJECT            :organizationName
 95:l=   5 prim:      PRINTABLESTRING   :sipit
102:l=  41 cons:    SET
104:l=  39 cons:     SEQUENCE
106:l=   3 prim:      OBJECT            :organizationalUnitName
111:l=  32 prim:      PRINTABLESTRING   :
                      Sipit Test Certificate Authority
145:l=  30 cons:   SEQUENCE
147:l=  13 prim:    UTCTIME           :030718122152Z
162:l=  13 prim:    UTCTIME           :130715122152Z
177:l= 112 cons:   SEQUENCE
179:l=  11 cons:    SET
181:l=   9 cons:     SEQUENCE
183:l=   3 prim:      OBJECT            :countryName
188:l=   2 prim:      PRINTABLESTRING   :US
192:l=  19 cons:    SET
194:l=  17 cons:     SEQUENCE
196:l=   3 prim:      OBJECT            :stateOrProvinceName
201:l=  10 prim:      PRINTABLESTRING   :California
213:l=  17 cons:    SET
215:l=  15 cons:     SEQUENCE
217:l=   3 prim:      OBJECT            :localityName
222:l=   8 prim:      PRINTABLESTRING   :San Jose
232:l=  14 cons:    SET
234:l=  12 cons:     SEQUENCE
```

```
  236:l=   3 prim:      OBJECT            :organizationName
  241:l=   5 prim:      PRINTABLESTRING   :sipit
  248:l=  41 cons:     SET
  250:l=  39 cons:      SEQUENCE
  252:l=   3 prim:       OBJECT            :organizationalUnitName
  257:l=  32 prim:       PRINTABLESTRING   :
                         Sipit Test Certificate Authority
  291:l= 159 cons:   SEQUENCE
  294:l=  13 cons:    SEQUENCE
  296:l=   9 prim:     OBJECT            :rsaEncryption
  307:l=   0 prim:     NULL
  309:l= 141 prim:    BIT STRING
00 30 81 89 02 81 81 00-c3 22 1e 83 91 c5 03 2c   .0.......".....,
3c 8a f4 11 14 c6 4b 9d-fa 72 78 c6 b0 95 18 a7   <.....K..rx.....
e0 8c 79 ba 5d a4 ae 1e-21 2d 9d f1 0b 1c cf bd   ..y.]...!-......
5b 29 b3 90 13 73 66 92-6e df 4c b3 b3 1c 1f 2a   [)...sf.n.L....*
82 0a ba 07 4d 52 b0 f8-37 7b e2 0a 27 30 70 dd   ....MR..7{..'0p.
f9 2e 03 ff 2a 76 cd df-87 1a bd 71 eb e1 99 6a   ....*v.....q...j
c4 7f 8e 74 a0 77 85 04-e9 41 ad fc 03 b6 17 75   ...t.w...A.....u
aa 33 ea 0a 16 d9 fb 79-32 2e f8 cf 4d c6 34 a3   .3.....y2...M.4.
ff 1b d0 68 28 e1 9d e5-02 03 01 00 01            ...h(........
  453:l= 205 cons:   cont [ 3 ]
  456:l= 202 cons:    SEQUENCE
  459:l=  29 cons:     SEQUENCE
  461:l=   3 prim:      OBJECT    :X509v3 Subject Key Identifier
  466:l=  22 prim:      OCTET STRING
04 14 6b 46 17 14 ea 94-76 25 80 54 6e 13 54 da   ..kF....v%.Tn.T.
a1 e3 54 14 a1 b6                                 ..T...
  490:l= 154 cons:     SEQUENCE
  493:l=   3 prim:      OBJECT    :X509v3 Authority Key Identifier
  498:l= 146 prim:      OCTET STRING
30 81 8f 80 14 6b 46 17-14 ea 94 76 25 80 54 6e   0....kF....v%.Tn
13 54 da a1 e3 54 14 a1-b6 a1 74 a4 72 30 70 31   .T...T....t.r0p1
0b 30 09 06 03 55 04 06-13 02 55 53 31 13 30 11   .0...U....US1.0.
06 03 55 04 08 13 0a 43-61 6c 69 66 6f 72 6e 69   ..U....Californi
61 31 11 30 0f 06 03 55-04 07 13 08 53 61 6e 20   a1.0...U....San
4a 6f 73 65 31 0e 30 0c-06 03 55 04 0a 13 05 73   Jose1.0...U....s
69 70 69 74 31 29 30 27-06 03 55 04 0b 13 20 53   ipit1)0'..U... S
69 70 69 74 20 54 65 73-74 20 43 65 72 74 69 66   ipit Test Certif
69 63 61 74 65 20 41 75-74 68 6f 72 69 74 79 82   icate Authority.
01                                                .
      0092 - <SPACES/NULS>
  647:l=  12 cons:     SEQUENCE
  649:l=   3 prim:      OBJECT            :X509v3 Basic Constraints
  654:l=   5 prim:      OCTET STRING
30 03 01 01 ff                                    0....
  661:l=  13 cons:  SEQUENCE
  663:l=   9 prim:   OBJECT            :sha1WithRSAEncryption
```

```
  674:l=   0 prim:   NULL
  676:l= 129 prim:  BIT STRING
00 96 6d 1b ef d5 91 93-45 7c 5b 1f cf c4 aa 47   ..m.....E|[....G
52 0b 34 a8 50 fa ec fa-b4 2a 47 4c 5d 41 a7 3d   R.4.P....*GL]A.=
c0 d6 3f 9e 56 5b 91 1d-ce a8 07 b3 1b a4 9f 9a   ..?.V[..........
49 6f 7f e0 ce 83 94 71-42 af fe 63 a2 34 dc b4   Io.....qB..c.4..
5e a5 ce ca 79 50 e9 6a-99 4c 14 69 e9 7c ab 22   ^...yP.j.L.i.|."
6c 44 cc 8a 9c 33 6b 23-50 42 05 1f e1 c2 81 88   lD...3k#PB......
5f ba e5 47 bb 85 9b 83-25 ad 84 32 ff 2a 5b 8b   _..G....%..2.*[.
70 12 11 83 61 c9 69 15-4f 58 a3 3c 92 d4 e8 6f   p...a.i.OX.<...o
52                                                R
```

3.2 Host Certificate

The certificate for the host example.com is shown below. Note that the Subject Alternative Name is set to example.com and is a DNS type. The certificates for the other hosts are shown in Appendix B.

```
   Data:
       Version: 3 (0x2)
       Serial Number:
           01:95:00:71:02:33:00:55
       Signature Algorithm: sha1WithRSAEncryption
       Issuer: C=US, ST=California, L=San Jose, O=sipit,
               OU=Sipit Test Certificate Authority
       Validity
           Not Before: Feb  3 18:49:08 2005 GMT
           Not After : Feb  3 18:49:08 2008 GMT
       Subject: C=US, ST=California, L=San Jose, O=sipit,
                CN=example.com
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:e6:31:76:b5:27:cc:8d:32:85:56:70:f7:c2:33:
                   33:32:26:42:5e:3c:68:71:7b:1f:79:50:d0:72:27:
                   3b:4a:af:f2:ce:d1:0c:bc:c0:5f:31:6a:43:e7:7c:
                   ad:64:bd:c7:e6:25:9f:aa:cd:2d:90:aa:68:84:62:
                   7b:05:be:43:a5:af:bb:ea:9d:a9:5b:a4:53:9d:22:
                   8b:da:96:2e:1f:3f:92:46:b8:cc:c8:24:3c:46:cd:
                   5d:2d:64:85:b1:a4:ca:01:f1:8e:c5:7e:0f:ff:00:
                   91:a3:ea:cb:3e:12:02:75:a4:bb:08:c8:d0:2a:ef:
                   b3:bb:72:7a:98:e5:ff:9f:81
               Exponent: 65537 (0x10001)
       X509v3 extensions:
           X509v3 Subject Alternative Name:
               DNS:example.com
           X509v3 Basic Constraints:
               CA:FALSE
           X509v3 Subject Key Identifier:
               22:EA:CB:38:66:1D:F1:96:0C:9A:47:B6:BB:1C:52:
               44:B0:77:65:8D
   Signature Algorithm: sha1WithRSAEncryption
       ae:eb:49:ed:1e:f1:8d:26:a9:6d:03:82:92:d5:df:44:c4:1e:
       1f:07:75:88:37:e4:76:97:35:12:59:98:79:78:16:6e:3b:b1:
       c0:2b:db:85:02:6b:74:c9:5b:19:92:da:7e:f5:41:0b:bc:d2:
       dd:45:aa:6f:be:24:dc:48:57:66:d9:2e:82:df:9e:8d:70:03:
       73:75:ef:8f:7a:56:4c:cc:42:bd:31:45:b0:5e:ff:d1:3b:c4:
       82:ee:fd:a7:c1:10:34:eb:81:49:1a:6b:86:7e:c7:61:1d:b3:
       b9:0a:02:bd:84:f8:47:af:cf:f1:a8:73:a8:31:1d:20:7a:06:
       7f:ac
```

3.3 User Certificates

The user certificate for fluffy@example.com is shown below. Note

that the Subject Alternative Name has a list of names with different URL types such as a sip, im, or pres URL. This is necessary for interoperating with CPIM gateway. In this example, example.com is the domain for fluffy, the message could be coming from a host called example.com, and the AOR in the user certificate would still be the same. The others are shown in Appendix B.

```
   Data:
       Version: 3 (0x2)
       Serial Number:
           01:95:00:71:02:33:00:58
       Signature Algorithm: sha1WithRSAEncryption
       Issuer: C=US, ST=California, L=San Jose, O=sipit,
               OU=Sipit Test Certificate Authority
       Validity
           Not Before: Feb  3 18:49:34 2005 GMT
           Not After : Feb  3 18:49:34 2008 GMT
       Subject: C=US, ST=California, L=San Jose, O=sipit,
                CN=fluffy@example.com
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:ca:ab:9b:9b:4e:3c:d5:45:3c:ce:00:a6:36:a8:
                   b9:ec:d2:76:e2:b9:9b:e8:28:aa:ba:86:22:c5:cf:
                   33:3e:4f:6d:56:21:ae:bd:54:84:7c:14:14:f9:7d:
                   99:85:00:4e:93:d6:fd:6b:d4:d1:d4:55:8e:c9:89:
                   b1:af:2b:5f:23:99:4a:95:e5:68:65:64:1d:12:a7:
                   db:d3:d5:97:18:47:35:9c:e6:88:27:9d:a8:6c:ca:
                   2a:84:e6:62:d8:f1:e9:a2:1a:39:7e:0e:0f:90:a5:
                   a6:79:21:bc:2a:67:b4:dd:69:90:82:9a:ae:1f:02:
                   52:8a:58:d3:f5:d0:d4:66:67
               Exponent: 65537 (0x10001)
       X509v3 extensions:
           X509v3 Subject Alternative Name:
               URI:sip:fluffy@example.com, URI:im:fluffy@example.com,
               URI:pres:fluffy@example.com
           X509v3 Basic Constraints:
               CA:FALSE
           X509v3 Subject Key Identifier:
               EC:DA:98:5E:E9:F7:F7:D7:EC:2B:29:4B:DA:25:EE:C7:C7:
               7E:95:70
   Signature Algorithm: sha1WithRSAEncryption
       4c:46:49:6e:01:48:e2:d4:6e:d7:48:a1:f3:7b:c8:a5:98:37:
       a5:44:46:58:9f:4a:37:7d:90:fb:5f:ff:36:bd:67:31:f0:29:
       de:0a:e2:ea:b9:f0:5c:9f:ad:a0:de:e5:4e:42:8f:11:d8:41:
       ea:68:be:db:c2:1e:fa:e5:8a:2d:7f:66:13:29:e9:da:8f:fb:
       80:bf:7e:5e:b6:04:ad:08:5e:58:95:b7:c5:38:85:d5:65:31:
       ad:80:cb:28:a7:4c:ad:11:fd:41:3b:37:77:5a:de:85:96:3d:
       66:eb:5f:9a:f8:60:5f:8e:b1:fc:4a:43:53:b6:11:4d:2e:f4:
       3d:ff
```

4. Callflow with Message Over TLS

4.1 TLS with Server Authentication

The flow below shows the edited SSLDump output of the host example.com forming a TLS connection to example.net. In this example mutual authentication is not used. Note that the client proposed three protocol suites including the required TLS_RSA_WITH_AES_128_CBC_SHA. The certificate returned by the server contains a Subject Alternative Name that is set to example.net. A detailed discussion of TLS can be found in [11].

```
New TCP connection #1: 127.0.0.1(55768) <-> 127.0.0.1(5061)
1 1  0.0060 (0.0060)  C>SV3.1(49)  Handshake
      ClientHello
        Version 3.1
        random[32]=
          42 16 8c c7 82 cd c5 87 42 ba f5 1c 91 04 fb 7d
          4d 6c 56 f1 db 1d ce 8a b1 25 71 5a 68 01 a2 14
        cipher suites
        TLS_RSA_WITH_AES_256_CBC_SHA
        TLS_RSA_WITH_AES_128_CBC_SHA
        TLS_RSA_WITH_3DES_EDE_CBC_SHA
        compression methods
                  NULL
1 2  0.0138 (0.0077)  S>CV3.1(74)  Handshake
      ServerHello
        Version 3.1
        random[32]=
          42 16 8c c7 c9 2c 43 42 bb 69 a5 ba f1 2d 69 75
          c3 8d 3a 85 78 19 f2 e4 d9 2b 72 b4 cc dd e4 72
        session_id[32]=
          06 37 e9 22 56 29 e6 b4 3a 6e 53 fe 56 27 ed 1f
          2a 75 34 65 f0 91 fc 79 cf 90 da ac f4 6f 64 b5
        cipherSuite         TLS_RSA_WITH_AES_256_CBC_SHA
        compressionMethod                   NULL
1 3  0.0138 (0.0000)  S>CV3.1(1477)  Handshake
      Certificate
1 4  0.0138 (0.0000)  S>CV3.1(4)  Handshake
      ServerHelloDone
1 5  0.0183 (0.0045)  C>SV3.1(134)  Handshake
      ClientKeyExchange
        EncryptedPreMasterSecret[128]=
          a6 bd d9 4b 76 4b 9d 6f 7b 12 8a e4 52 75 9d 74
          4f 06 e4 b0 bc 69 96 d7 42 ba 77 01 b6 9e 64 b0
          ea c5 aa de 59 41 e4 f3 9e 1c 1c a9 48 f5 0a 3f
          5e c3 50 23 15 d7 46 1d 69 79 76 ba 5e c8 ac 39
```

```
            23 71 d0 0c 18 a6 a9 77 0f 7d 49 61 ef 6f 8d 32
            54 f5 a4 1d 19 33 0a 64 ee 56 91 9b f4 f7 50 b1
            11 4b 81 46 4c 36 df 70 98 04 dc 5c 8a 16 a9 2e
            58 67 ae 5e 7a a9 44 2b 0b 7c 9c 2f 16 25 1a e9
   1 6  0.0183 (0.0000)  C>SV3.1(1)  ChangeCipherSpec
   1 7  0.0183 (0.0000)  C>SV3.1(48)  Handshake
   1 8  0.0630 (0.0447)  S>CV3.1(1)  ChangeCipherSpec
   1 9  0.0630 (0.0000)  S>CV3.1(48)  Handshake
   1 10 0.3274 (0.2643)  C>SV3.1(32)  application_data
   1 11 0.3274 (0.0000)  C>SV3.1(720)  application_data
   1 12 0.3324 (0.0050)  S>CV3.1(32)  application_data
   1 13 0.3324 (0.0000)  S>CV3.1(384)  application_data
   1    9.2491 (8.9166)  C>S  TCP FIN
   1    9.4023 (0.1531)  S>C  TCP FIN
```

4.2 MESSAGE Message Over TLS

Once the TLS session is set up, the following MESSAGE message is sent from fluffy@example.com to kumiko@example.net. Note that the URI has a SIPS URL and that the VIA indicates that TLS was used. In order to format this document, it was necessary to break up some of the lines across continuation lines but the original messages have no continuations lines and no breaks in the Identity header field value.

```
MESSAGE sips:kumiko@example.net SIP/2.0
To: <sips:kumiko@example.net>
From: <sips:fluffy@example.com>;tag=03de46e1
Via: SIP/2.0/TLS 127.0.0.1:5071;
     branch=z9hG4bK-d87543-58c826887160f95f-1--d87543-;rport
Call-ID: 0dc68373623af98a@Y2ouY2lzY28uc2lwaXQubmV0
CSeq: 1 MESSAGE
Contact: <sips:fluffy@127.0.0.1:5071;transport=TLS>
Max-Forwards: 70
Content-Transfer-Encoding: binary
Content-Type: text/plain
Date: Sat, 19 Feb 2005 00:48:07 GMT
User-Agent: SIPimp.org/0.2.5 (curses)
Identity: qKUEWvgss+F0pQHJCyarb8IMbDh1d1gi1Aq51ty61bO+ug5ZQzo31xn
          MAFHUe0tzNVoyOfmGUY2dIEWJ2iZlGI5EW3RF5hGN9f0y39iCRqGEAE
          B4UG5ocU4RzgXfK3Durle/66rkyCaLPJQ/pzgA+qW/nQytSuzewhDrD
          FRrCBQ=
Content-Length: 6

Hello!
```

The response is sent from example.net to example.com over the same

TLS connection. It is shown below.

```
SIP/2.0 200 OK
To: <sips:kumiko@example.net>;tag=4c53f1b8
From: <sips:fluffy@example.com>;tag=03de46e1
Via: SIP/2.0/TLS 127.0.0.1:5071;
     branch=z9hG4bK-d87543-58c826887160f95f-1--d87543-;
     rport=55768;received=127.0.0.1
Call-ID: 0dc68373623af98a@Y2ouY2lzY28uc2lwaXQubmV0
CSeq: 1 MESSAGE
Contact: <sips:kumiko@127.0.0.1:5061;transport=TLS>
Content-Length: 0
```

5. Callflow with S/MIME-secured Message

5.1 MESSAGE Message with Signed Body

Example Signed Message. The value on the Content-Type line has been broken across lines to fit on the page but it should not broken across lines in actual implementations.

```
MESSAGE sip:kumiko@example.net SIP/2.0
To: <sip:kumiko@example.net>
From: <sip:fluffy@example.com>;tag=0c523b42
Via: SIP/2.0/UDP 68.122.119.3:5060;
     branch=z9hG4bK-d87543-16a1192b7960f635-1--d87543-;rport
Call-ID: 27bb7608596d8914@Y2ouY2lzY28uc2lwaXQubmV0
CSeq: 1 MESSAGE
Contact: <sip:fluffy@68.122.119.3:5060>
Max-Forwards: 70
Content-Transfer-Encoding: binary
Content-Type: multipart/signed;boundary=151aa2144df0f6bd;\
              micalg=sha1;protocol=application/pkcs7-signature
Date: Fri, 04 Feb 2005 20:17:12 GMT
User-Agent: SIPimp.org/0.2.5 (curses)
Content-Length: 1544

--151aa2144df0f6bd
Content-Type: text/plain
Content-Transfer-Encoding: binary

Hello
--151aa2144df0f6bd
Content-Type: application/pkcs7-mime;name=smime.p7s
Content-Disposition: attachment;handling=required;filename=smime.p7s
Content-Transfer-Encoding: binary

******************
* BINARY BLOB 1   *
******************
--151aa2144df0f6bd--
```

It is important to note that the signature is computed across includes the header and excludes the boundary. The value on the Message-body line ends with CRLF. The CRLF is included in the boundary and should not be shown.

```
Content-Type: text/plain
Content-Transfer-Encoding: binary

Hello
```

ASN.1 parse of binary Blob 1. Note that at address 30, the hash for the signature is specified as SHA1. Also note that from address 52 to 777, the sender's certificate is attached, although it is optional [8].

```
   0: SEQUENCE {
```

```
  4:   OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)
 15:   [0] {
 19:     SEQUENCE {
 23:       INTEGER 1
 26:       SET {
 28:         SEQUENCE {
 30:           OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
 37:           NULL
   :           }
   :         }
 39:       SEQUENCE {
 41:        OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
   :        }
 52:      [0] {
 56:        SEQUENCE {
 60:          SEQUENCE {
 64:            [0] {
 66:              INTEGER 2
   :              }
 69:            INTEGER 01 95 00 71 02 33 00 58
 79:            SEQUENCE {
 81:              OBJECT IDENTIFIER
   :                sha1withRSAEncryption (1 2 840 113549 1 1 5)
 92:              NULL
   :              }
 94:            SEQUENCE {
 96:              SET {
 98:                SEQUENCE {
100:                  OBJECT IDENTIFIER countryName (2 5 4 6)
105:                  PrintableString 'US'
   :                  }
   :                }
109:              SET {
111:                SEQUENCE {
113:                OBJECT IDENTIFIER stateOrProvinceName (2 5 4 8)
118:                PrintableString 'California'
   :                }
   :                }
130:              SET {
132:                SEQUENCE {
134:                  OBJECT IDENTIFIER localityName (2 5 4 7)
139:                  PrintableString 'San Jose'
   :                  }
   :                }
149:              SET {
151:                SEQUENCE {
153:                  OBJECT IDENTIFIER organizationName (2 5 4 10)
158:                  PrintableString 'sipit'
```

```
    :                 }
    :               }
  165:              SET {
  167:               SEQUENCE {
  169:                OBJECT IDENTIFIER
    :                 organizationalUnitName (2 5 4 11)
  174:               PrintableString 'Sipit Test Certificate Authority'
    :                 }
    :                }
    :               }
  208:              SEQUENCE {
  210:                UTCTime 03/02/2005 18:49:34 GMT
  225:                UTCTime 03/02/2008 18:49:34 GMT
    :                }
  240:              SEQUENCE {
  242:                SET {
  244:                  SEQUENCE {
  246:                    OBJECT IDENTIFIER countryName (2 5 4 6)
  251:                    PrintableString 'US'
    :                    }
    :                  }
  255:                SET {
  257:                 SEQUENCE {
  259:                  OBJECT IDENTIFIER stateOrProvinceName (2 5 4 8)
  264:                    PrintableString 'California'
    :                    }
    :                  }
  276:                SET {
  278:                  SEQUENCE {
  280:                    OBJECT IDENTIFIER localityName (2 5 4 7)
  285:                    PrintableString 'San Jose'
    :                    }
    :                  }
  295:                SET {
  297:                  SEQUENCE {
  299:                    OBJECT IDENTIFIER organizationName (2 5 4 10)
  304:                    PrintableString 'sipit'
    :                    }
    :                  }
  311:                SET {
  313:                  SEQUENCE {
  315:                    OBJECT IDENTIFIER commonName (2 5 4 3)
  320:                    TeletexString 'fluffy@example.com'
    :                    }
    :                  }
    :                }
  340:              SEQUENCE {
  343:                SEQUENCE {
```

```
345:                OBJECT IDENTIFIER
   :                  rsaEncryption (1 2 840 113549 1 1 1)
356:                NULL
   :                }
358:              BIT STRING, encapsulates {
362:                SEQUENCE {
365:                  INTEGER
   :                 00 CA AB 9B 9B 4E 3C D5 45 3C CE 00 A6 36 A8 B9
   :                 EC D2 76 E2 B9 9B E8 28 AA BA 86 22 C5 CF 33 3E
   :                 4F 6D 56 21 AE BD 54 84 7C 14 14 F9 7D 99 85 00
   :                 4E 93 D6 FD 6B D4 D1 D4 55 8E C9 89 B1 AF 2B 5F
   :                 23 99 4A 95 E5 68 65 64 1D 12 A7 DB D3 D5 97 18
   :                 47 35 9C E6 88 27 9D A8 6C CA 2A 84 E6 62 D8 F1
   :                 E9 A2 1A 39 7E 0E 0F 90 A5 A6 79 21 BC 2A 67 B4
   :                 DD 69 90 82 9A AE 1F 02 52 8A 58 D3 F5 D0 D4 66
   :                        [ Another 1 bytes skipped ]
497:                  INTEGER 65537
   :                  }
   :                }
   :              }
502:            [3] {
504:              SEQUENCE {
506:                SEQUENCE {
508:                  OBJECT IDENTIFIER subjectAltName (2 5 29 17)
513:                  OCTET STRING, encapsulates {
515:                    SEQUENCE {
517:                      [6] 'sip:fluffy@example.com'
541:                      [6] 'im:fluffy@example.com'
564:                      [6] 'pres:fluffy@example.com'
   :                      }
   :                    }
   :                  }
589:                SEQUENCE {
591:                  OBJECT IDENTIFIER basicConstraints (2 5 29 19)
596:                  OCTET STRING, encapsulates {
598:                    SEQUENCE {}
   :                    }
   :                  }
600:                SEQUENCE {
602:                  OBJECT IDENTIFIER
                             subjectKeyIdentifier (2 5 29 14)
607:                  OCTET STRING, encapsulates {
609:                    OCTET STRING
   :                 EC DA 98 5E E9 F7 F7 D7 EC 2B 29 4B DA 25 EE C7
   :                 C7 7E 95 70
   :                    }
   :                  }
   :                }
```

```
   :              }
   :            }
631:           SEQUENCE {
633:             OBJECT IDENTIFIER
   :               sha1withRSAEncryption (1 2 840 113549 1 1 5)
644:             NULL
   :             }
646:           BIT STRING
   :             4C 46 49 6E 01 48 E2 D4 6E D7 48 A1 F3 7B C8 A5
   :             98 37 A5 44 46 58 9F 4A 37 7D 90 FB 5F FF 36 BD
   :             67 31 F0 29 DE 0A E2 EA B9 F0 5C 9F AD A0 DE E5
   :             4E 42 8F 11 D8 41 EA 68 BE DB C2 1E FA E5 8A 2D
   :             7F 66 13 29 E9 DA 8F FB 80 BF 7E 5E B6 04 AD 08
   :             5E 58 95 B7 C5 38 85 D5 65 31 AD 80 CB 28 A7 4C
   :             AD 11 FD 41 3B 37 77 5A DE 85 96 3D 66 EB 5F 9A
   :             F8 60 5F 8E B1 FC 4A 43 53 B6 11 4D 2E F4 3D FF
   :           }
   :         }
778:       SET {
782:         SEQUENCE {
786:           INTEGER 1
789:           SEQUENCE {
791:             SEQUENCE {
793:               SET {
795:                 SEQUENCE {
797:                   OBJECT IDENTIFIER countryName (2 5 4 6)
802:                   PrintableString 'US'
   :                   }
   :                 }
806:               SET {
808:                 SEQUENCE {
810:                   OBJECT IDENTIFIER
                              stateOrProvinceName (2 5 4 8)
815:                   PrintableString 'California'
   :                   }
   :                 }
827:               SET {
829:                 SEQUENCE {
831:                   OBJECT IDENTIFIER localityName (2 5 4 7)
836:                   PrintableString 'San Jose'
   :                   }
   :                 }
846:               SET {
848:                 SEQUENCE {
850:                   OBJECT IDENTIFIER organizationName (2 5 4 10)
855:                   PrintableString 'sipit'
   :                   }
   :                 }
```

```
    862:              SET {
    864:                SEQUENCE {
    866:                  OBJECT IDENTIFIER
       :                    organizationalUnitName (2 5 4 11)
    871:                  PrintableString
                            'Sipit Test Certificate Authority'
       :                  }
       :                }
       :              }
    905:            INTEGER 01 95 00 71 02 33 00 58
       :            }
    915:          SEQUENCE {
    917:            OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
    924:            NULL
       :            }
    926:          [0] {
    929:            SEQUENCE {
    931:              OBJECT IDENTIFIER
                            contentType (1 2 840 113549 1 9 3)
    942:              SET {
    944:                OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
       :                }
       :              }
    955:            SEQUENCE {
    957:              OBJECT IDENTIFIER
                            signingTime (1 2 840 113549 1 9 5)
    968:              SET {
    970:                UTCTime 04/02/2005 20:17:12 GMT
       :                }
       :              }
    985:            SEQUENCE {
    987:              OBJECT IDENTIFIER
                            messageDigest (1 2 840 113549 1 9 4)
    998:              SET {
   1000:                OCTET STRING
       :                DA 23 80 0F 1E B9 E1 95 CC 7E 55 3D 49 AE C1 7A
       :                D5 99 DA 2B
       :                }
       :              }
   1022:              SEQUENCE {
   1024:                OBJECT IDENTIFIER
       :                  sMIMECapabilities (1 2 840 113549 1 9 15)
   1035:              SET {
   1037:                SEQUENCE {
   1039:                  SEQUENCE {
   1041:                    OBJECT IDENTIFIER
       :                      des-EDE3-CBC (1 2 840 113549 3 7)
       :                    }
```

```
   1051:                   SEQUENCE {
   1053:                     OBJECT IDENTIFIER
                                    rc2CBC (1 2 840 113549 3 2)
   1063:                     INTEGER 128
       :                     }
   1067:                   SEQUENCE {
   1069:                     OBJECT IDENTIFIER
                                    rc2CBC (1 2 840 113549 3 2)
   1079:                     INTEGER 64
       :                     }
   1082:                   SEQUENCE {
   1084:                     OBJECT IDENTIFIER
                                    desCBC (1 3 14 3 2 7)
       :                     }
   1091:                   SEQUENCE {
   1093:                     OBJECT IDENTIFIER
                                    rc2CBC (1 2 840 113549 3 2)
   1103:                     INTEGER 40
       :                     }
       :                   }
       :                 }
       :               }
       :             }
   1106:           SEQUENCE {
   1108:             OBJECT IDENTIFIER
                            rsaEncryption (1 2 840 113549 1 1 1)
   1119:             NULL
       :             }
   1121:           OCTET STRING
       :             66 F0 C9 C0 78 69 27 F9 81 05 05 F1 E1 54 B9 5C
       :             3A 2B 34 68 0E 31 19 06 DD 00 34 40 66 DF D8 2F
       :             0C BC 6C 80 A2 0B 45 5B 68 36 81 C1 F2 8C AF CA
       :             0E 9B 9E A0 BD BC 4E 47 2D 99 B6 76 3E F5 9E B7
       :             77 78 BB A4 40 35 DE 2E 26 CE AB DA 70 A7 65 BA
       :             89 51 E9 AB F1 26 CA 54 1C 05 4D 01 B0 AE 75 6A
       :             3F A3 2C 5D 4F A0 46 77 45 6D 11 DE 7C F1 0D C4
       :             61 10 67 D2 3D 56 B2 3E A5 C1 2F 6E 0D 5C 4D FC
       :           }
       :         }
       :       }
       :     }
       :   }
```

5.2 MESSAGE Message with Encrypted Body

Example encrypted message:

```
MESSAGE sip:kumiko@example.net SIP/2.0
To: <sip:kumiko@example.net>
From: <sip:fluffy@example.com>;tag=6d2a39e4
Via: SIP/2.0/UDP 68.122.119.3:5060;
     branch=z9hG4bK-d87543-44ddc0a217a51788-1--d87543-;rport
Call-ID: 031be67669ea9799@Y2ouY2lzY28uc2lwaXQubmV0
CSeq: 1 MESSAGE
Contact: <sip:fluffy@68.122.119.3:5060>
Max-Forwards: 70
Content-Disposition: attachment;handling=required;filename=smime.p7
Content-Transfer-Encoding: binary
Content-Type: application/pkcs7-mime;\
              smime-type=enveloped-data;name=smime.p7m
Date: Fri, 04 Feb 2005 20:04:10 GMT
User-Agent: SIPimp.org/0.2.5 (curses)
Content-Length: 418

*****************
* BINARY BLOB 2 *
*****************
```

ASN.1 parse of binary Blob 2. Note that at address 324, the encryption is set to des-ebe3-cbc.

```
   0: SEQUENCE {
   4:   OBJECT IDENTIFIER envelopedData (1 2 840 113549 1 7 3)
  15:   [0] {
  19:     SEQUENCE {
  23:       INTEGER 0
  26:       SET {
  30:         SEQUENCE {
  34:           INTEGER 0
  37:           SEQUENCE {
  39:             SEQUENCE {
  41:               SET {
  43:                 SEQUENCE {
  45:                   OBJECT IDENTIFIER countryName (2 5 4 6)
  50:                   PrintableString 'US'
    :                   }
    :                 }
  54:               SET {
  56:                 SEQUENCE {
  58:                   OBJECT IDENTIFIER
                               stateOrProvinceName (2 5 4 8)
  63:                   PrintableString 'California'
    :                   }
    :                 }
```

```
     75:               SET {
     77:                 SEQUENCE {
     79:                   OBJECT IDENTIFIER localityName (2 5 4 7)
     84:                   PrintableString 'San Jose'
      :                   }
      :                 }
     94:               SET {
     96:                 SEQUENCE {
     98:                  OBJECT IDENTIFIER organizationName (2 5 4 10)
    103:                  PrintableString 'sipit'
      :                   }
      :                 }
    110:               SET {
    112:                 SEQUENCE {
    114:                   OBJECT IDENTIFIER
      :                     organizationalUnitName (2 5 4 11)
    119:                   PrintableString
                             'Sipit Test Certificate Authority'
      :                   }
      :                 }
      :               }
    153:             INTEGER 01 95 00 71 02 33 00 57
      :             }
    163:           SEQUENCE {
    165:             OBJECT IDENTIFIER
                            rsaEncryption (1 2 840 113549 1 1 1)
    176:             NULL
      :             }
    178:           OCTET STRING
      :             BC 8A DF 69 69 F9 72 2A 13 32 62 DF FA 83 FE EF
      :             28 6A 3A 63 75 FC 2F 83 93 13 21 A0 62 FC 29 01
      :             35 F7 81 B2 3B 2E FD F8 E4 D3 DD E0 C3 52 32 13
      :             B0 37 31 9D 5C A0 41 3A C5 A5 95 C9 95 08 DA 47
      :             E9 1D 72 F8 75 71 B0 05 E0 0A B3 33 60 F2 9C 0B
      :             CF FB DE 2E BC 1B C5 8F AE 5F BB 9E 73 21 E2 E2
      :             2F 34 B7 2F F0 BB B8 94 76 8F 6D 4B 9A 7F CD 4E
      :             4A 01 0D 12 ED 70 81 00 8C B8 37 9E 6B 80 66 03
      :           }
      :         }
    309:       SEQUENCE {
    311:         OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
    322:         SEQUENCE {
    324:           OBJECT IDENTIFIER des-EDE3-CBC (1 2 840 113549 3 7)
    334:           OCTET STRING 05 8B C4 DC 50 5E D7 09
      :           }
    344:         [0]
      :           60 23 E0 B9 79 CC 39 5B 86 E9 87 8C C2 C6 A0 EE
      :           7A 15 3F 0A BB D8 F5 6C EF 4D 18 52 C1 25 65 F5
```

```
:           84 5F C7 1C 78 52 1D 33 37 2B 41 69 52 D0 7C FD
:           67 A2 2E 96 2E AA 8F 6F 66 F2 9E 2F 74 12 A7 C7
:           CC 9E 83 D1 D9 C4 57 A3
:         }
:       }
:     }
:   }
```

5.3 MESSAGE Message with Encrypted and Signed Body

In the example below, one of the headers is contained in a box and is split across two lines. This was only done to make it fit in the RFC format. This header should not have the box around it and should be on one line with no whitespace between the "mime;" and the "smime-type". Note that Content-Type is split across lines for formatting but is not split in the real message.

```
MESSAGE sip:kumiko@example.net SIP/2.0
To: <sip:kumiko@example.net>
From: <sip:fluffy@example.com>;tag=361300da
Via: SIP/2.0/UDP 68.122.119.3:5060;
     branch=z9hG4bK-d87543-0710dbfb18ebb8e6-1--d87543-;rport
Call-ID: 5eda27a67de6283d@Y2ouY2lzY28uc2lwaXQubmV0
CSeq: 1 MESSAGE
Contact: <sip:fluffy@68.122.119.3:5060>
Max-Forwards: 70
Content-Transfer-Encoding: binary
Content-Type: multipart/signed;boundary=1af019eb7754ddf7;\
              micalg=sha1;protocol=application/pkcs7-signature
Date: Fri, 04 Feb 2005 20:07:14 GMT
User-Agent: SIPimp.org/0.2.5 (curses)
Content-Length: 2079

--1af019eb7754ddf7
|--See note about stuff in this box --------------------|
|Content-Type: application/pkcs7-mime;                  |
|              smime-type=enveloped-data;name=smime.p7m |
|-------------------------------------------------------|
Content-Disposition: attachment;handling=required;filename=smime.p7
Content-Transfer-Encoding: binary

*****************
* BINARY BLOB 3 *
*****************
--1af019eb7754ddf7
Content-Type: application/pkcs7-mime;name=smime.p7s
Content-Disposition: attachment;handling=required;filename=smime.p7s
Content-Transfer-Encoding: binary

*****************
* BINARY BLOB 4 *
*****************
--1af019eb7754ddf7--
```

Binary blob 3

```
   0: SEQUENCE {
   4:   OBJECT IDENTIFIER envelopedData (1 2 840 113549 1 7 3)
  15:   [0] {
  19:     SEQUENCE {
  23:       INTEGER 0
  26:       SET {
  30:         SEQUENCE {
  34:           INTEGER 0
```

```
     37:           SEQUENCE {
     39:             SEQUENCE {
     41:               SET {
     43:                 SEQUENCE {
     45:                   OBJECT IDENTIFIER countryName (2 5 4 6)
     50:                   PrintableString 'US'
      :                   }
      :                 }
     54:               SET {
     56:                 SEQUENCE {
     58:                   OBJECT IDENTIFIER
                                 stateOrProvinceName (2 5 4 8)
     63:                   PrintableString 'California'
      :                   }
      :                 }
     75:               SET {
     77:                 SEQUENCE {
     79:                   OBJECT IDENTIFIER localityName (2 5 4 7)
     84:                   PrintableString 'San Jose'
      :                   }
      :                 }
     94:               SET {
     96:                 SEQUENCE {
     98:                  OBJECT IDENTIFIER organizationName (2 5 4 10)
    103:                  PrintableString 'sipit'
      :                  }
      :                 }
    110:               SET {
    112:                 SEQUENCE {
    114:                   OBJECT IDENTIFIER
      :                     organizationalUnitName (2 5 4 11)
    119:                   PrintableString
                             'Sipit Test Certificate Authority'
      :                   }
      :                 }
      :               }
    153:             INTEGER 01 95 00 71 02 33 00 57
      :             }
    163:           SEQUENCE {
    165:             OBJECT IDENTIFIER
                            rsaEncryption (1 2 840 113549 1 1 1)
    176:             NULL
      :             }
    178:           OCTET STRING
      :             0D 65 F7 54 9B A6 A5 42 2B 12 0E AC 70 16 20 52
      :             64 22 B8 61 24 DD 88 38 AA 59 B6 55 D9 73 79 B8
      :             7B 10 A9 13 1C A3 A4 00 CE F7 0A 81 80 5B 37 E3
      :             2E A4 16 58 43 DF A1 FF 8A FD 43 1C 52 D5 79 43
```

```
      :             79 7F 7A FF CB 09 49 0F 2A 49 12 6C EC C5 58 0F
      :             4F 02 75 47 12 8C 8C 8F 84 49 FC 19 F0 24 9F F4
      :             7A 22 53 64 92 40 CA 03 41 3B 22 B7 E2 8A B5 79
      :             22 22 9F BC 10 65 0D 69 02 1C 51 35 6D A2 9D 77
      :           }
      :         }
    309:       SEQUENCE {
    311:         OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
    322:         SEQUENCE {
    324:           OBJECT IDENTIFIER des-EDE3-CBC (1 2 840 113549 3 7)
    334:           OCTET STRING E2 6A CB 3E F3 BC A2 00
      :           }
    344:         [0]
      :           62 45 2E 76 6F 99 83 F2 C5 0B 9C 87 9E 66 C5 38
      :           F1 57 68 5F CF F1 AF 44 5E 02 84 FF C3 76 94 D4
      :           9C 34 6B AD 2E 4A 1A 57 4B 88 4C A7 55 7C BF AB
      :           BB FD 15 E6 20 ED 22 36 73 2E 61 B5 69 37 A8 0C
      :           43 D1 A1 02 0E B9 B5 69
      :         }
      :       }
      :     }
      :   }
```

Binary Blob 4

```
     0: SEQUENCE {
     4:   OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)
    15:   [0] {
    19:     SEQUENCE {
    23:       INTEGER 1
    26:       SET {
    28:         SEQUENCE {
    30:           OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
    37:           NULL
      :           }
      :         }
    39:       SEQUENCE {
    41:         OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
      :         }
    52:       [0] {
    56:         SEQUENCE {
    60:           SEQUENCE {
    64:             [0] {
    66:               INTEGER 2
      :               }
    69:             INTEGER 01 95 00 71 02 33 00 58
    79:             SEQUENCE {
```

```
     81:               OBJECT IDENTIFIER
      :                 sha1withRSAEncryption (1 2 840 113549 1 1 5)
     92:               NULL
      :               }
     94:             SEQUENCE {
     96:               SET {
     98:                 SEQUENCE {
    100:                   OBJECT IDENTIFIER countryName (2 5 4 6)
    105:                   PrintableString 'US'
      :                   }
      :                 }
    109:               SET {
    111:                SEQUENCE {
    113:                 OBJECT IDENTIFIER stateOrProvinceName (2 5 4 8)
    118:                 PrintableString 'California'
      :                 }
      :                }
    130:               SET {
    132:                 SEQUENCE {
    134:                   OBJECT IDENTIFIER localityName (2 5 4 7)
    139:                   PrintableString 'San Jose'
      :                   }
      :                 }
    149:               SET {
    151:                 SEQUENCE {
    153:                   OBJECT IDENTIFIER organizationName (2 5 4 10)
    158:                   PrintableString 'sipit'
      :                   }
      :                 }
    165:               SET {
    167:                 SEQUENCE {
    169:                   OBJECT IDENTIFIER
      :                     organizationalUnitName (2 5 4 11)
    174:                   PrintableString
                            'Sipit Test Certificate Authority'
      :                   }
      :                 }
      :               }
    208:             SEQUENCE {
    210:               UTCTime 03/02/2005 18:49:34 GMT
    225:               UTCTime 03/02/2008 18:49:34 GMT
      :               }
    240:             SEQUENCE {
    242:               SET {
    244:                 SEQUENCE {
    246:                   OBJECT IDENTIFIER countryName (2 5 4 6)
    251:                   PrintableString 'US'
      :                   }
```

```
    :                 }
  255:              SET {
  257:                SEQUENCE {
  259:                  OBJECT IDENTIFIER
                                stateOrProvinceName (2 5 4 8)
  264:                  PrintableString 'California'
    :                   }
    :                 }
  276:              SET {
  278:                SEQUENCE {
  280:                  OBJECT IDENTIFIER localityName (2 5 4 7)
  285:                  PrintableString 'San Jose'
    :                   }
    :                 }
  295:              SET {
  297:                SEQUENCE {
  299:                 OBJECT IDENTIFIER organizationName (2 5 4 10)
  304:                 PrintableString 'sipit'
    :                  }
    :                 }
  311:              SET {
  313:                SEQUENCE {
  315:                  OBJECT IDENTIFIER commonName (2 5 4 3)
  320:                  TeletexString 'fluffy@example.com'
    :                   }
    :                 }
    :               }
  340:             SEQUENCE {
  343:               SEQUENCE {
  345:                 OBJECT IDENTIFIER
    :                   rsaEncryption (1 2 840 113549 1 1 1)
  356:                 NULL
    :                  }
  358:               BIT STRING, encapsulates {
  362:                 SEQUENCE {
  365:                   INTEGER
    :                   00 CA AB 9B 9B 4E 3C D5 45 3C CE 00 A6 36 A8 B9
    :                   EC D2 76 E2 B9 9B E8 28 AA BA 86 22 C5 CF 33 3E
    :                   4F 6D 56 21 AE BD 54 84 7C 14 14 F9 7D 99 85 00
    :                   4E 93 D6 FD 6B D4 D1 D4 55 8E C9 89 B1 AF 2B 5F
    :                   23 99 4A 95 E5 68 65 64 1D 12 A7 DB D3 D5 97 18
    :                   47 35 9C E6 88 27 9D A8 6C CA 2A 84 E6 62 D8 F1
    :                   E9 A2 1A 39 7E 0E 0F 90 A5 A6 79 21 BC 2A 67 B4
    :                   DD 69 90 82 9A AE 1F 02 52 8A 58 D3 F5 D0 D4 66
    :                           [ Another 1 bytes skipped ]
  497:                   INTEGER 65537
    :                    }
    :                 }
```

```
   :              }
 502:             [3] {
 504:               SEQUENCE {
 506:                 SEQUENCE {
 508:                   OBJECT IDENTIFIER subjectAltName (2 5 29 17)
 513:                   OCTET STRING, encapsulates {
 515:                     SEQUENCE {
 517:                       [6] 'sip:fluffy@example.com'
 541:                       [6] 'im:fluffy@example.com'
 564:                       [6] 'pres:fluffy@example.com'
   :                       }
   :                     }
   :                   }
 589:                 SEQUENCE {
 591:                   OBJECT IDENTIFIER
                               basicConstraints (2 5 29 19)
 596:                   OCTET STRING, encapsulates {
 598:                     SEQUENCE {}
   :                     }
   :                   }
 600:                 SEQUENCE {
 602:                   OBJECT IDENTIFIER
                               subjectKeyIdentifier (2 5 29 14)
 607:                   OCTET STRING, encapsulates {
 609:                     OCTET STRING
   :                 EC DA 98 5E E9 F7 F7 D7 EC 2B 29 4B DA 25 EE C7
   :                 C7 7E 95 70
   :                     }
   :                   }
   :                 }
   :               }
   :             }
 631:           SEQUENCE {
 633:             OBJECT IDENTIFIER
   :               sha1withRSAEncryption (1 2 840 113549 1 1 5)
 644:             NULL
   :             }
 646:           BIT STRING
   :             4C 46 49 6E 01 48 E2 D4 6E D7 48 A1 F3 7B C8 A5
   :             98 37 A5 44 46 58 9F 4A 37 7D 90 FB 5F FF 36 BD
   :             67 31 F0 29 DE 0A E2 EA B9 F0 5C 9F AD A0 DE E5
   :             4E 42 8F 11 D8 41 EA 68 BE DB C2 1E FA E5 8A 2D
   :             7F 66 13 29 E9 DA 8F FB 80 BF 7E 5E B6 04 AD 08
   :             5E 58 95 B7 C5 38 85 D5 65 31 AD 80 CB 28 A7 4C
   :             AD 11 FD 41 3B 37 77 5A DE 85 96 3D 66 EB 5F 9A
   :             F8 60 5F 8E B1 FC 4A 43 53 B6 11 4D 2E F4 3D FF
   :           }
   :         }
```

```
    778:       SET {
    782:         SEQUENCE {
    786:           INTEGER 1
    789:           SEQUENCE {
    791:             SEQUENCE {
    793:               SET {
    795:                 SEQUENCE {
    797:                   OBJECT IDENTIFIER countryName (2 5 4 6)
    802:                   PrintableString 'US'
       :                   }
       :                 }
    806:               SET {
    808:                 SEQUENCE {
    810:                   OBJECT IDENTIFIER
                                  stateOrProvinceName (2 5 4 8)
    815:                   PrintableString 'California'
       :                   }
       :                 }
    827:               SET {
    829:                 SEQUENCE {
    831:                   OBJECT IDENTIFIER localityName (2 5 4 7)
    836:                   PrintableString 'San Jose'
       :                   }
       :                 }
    846:               SET {
    848:                 SEQUENCE {
    850:                  OBJECT IDENTIFIER organizationName (2 5 4 10)
    855:                  PrintableString 'sipit'
       :                  }
       :                 }
    862:               SET {
    864:                 SEQUENCE {
    866:                   OBJECT IDENTIFIER
       :                     organizationalUnitName (2 5 4 11)
    871:                   PrintableString
                            'Sipit Test Certificate Authority'
       :                   }
       :                 }
       :               }
    905:             INTEGER 01 95 00 71 02 33 00 58
       :             }
    915:           SEQUENCE {
    917:             OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
    924:             NULL
       :             }
    926:           [0] {
    929:             SEQUENCE {
    931:               OBJECT IDENTIFIER
```

```
                          contentType (1 2 840 113549 1 9 3)
    942:                SET {
    944:                  OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
        :                  }
        :                }
    955:              SEQUENCE {
    957:                OBJECT IDENTIFIER
                              signingTime (1 2 840 113549 1 9 5)
    968:                SET {
    970:                  UTCTime 04/02/2005 20:07:14 GMT
        :                  }
        :                }
    985:              SEQUENCE {
    987:                OBJECT IDENTIFIER
                              messageDigest (1 2 840 113549 1 9 4)
    998:                SET {
   1000:                  OCTET STRING
        :                  58 ED 12 DD 68 18 99 96 F9 4C 81 4C A6 51 BD 84
        :                  A8 BA F3 6A
        :                  }
        :                }
   1022:              SEQUENCE {
   1024:                OBJECT IDENTIFIER
        :                  sMIMECapabilities (1 2 840 113549 1 9 15)
   1035:                SET {
   1037:                  SEQUENCE {
   1039:                    SEQUENCE {
   1041:                      OBJECT IDENTIFIER
        :                       des-EDE3-CBC (1 2 840 113549 3 7)
        :                      }
   1051:                    SEQUENCE {
   1053:                      OBJECT IDENTIFIER
                                 rc2CBC (1 2 840 113549 3 2)
   1063:                      INTEGER 128
        :                      }
   1067:                    SEQUENCE {
   1069:                      OBJECT IDENTIFIER
                                 rc2CBC (1 2 840 113549 3 2)
   1079:                      INTEGER 64
        :                      }
   1082:                    SEQUENCE {
   1084:                      OBJECT IDENTIFIER desCBC (1 3 14 3 2 7)
        :                      }
   1091:                    SEQUENCE {
   1093:                      OBJECT IDENTIFIER
                                 rc2CBC (1 2 840 113549 3 2)
   1103:                      INTEGER 40
        :                      }
```

```
      :                   }
      :                 }
      :               }
      :             }
      :             }
   1106:           SEQUENCE {
   1108:             OBJECT IDENTIFIER
                            rsaEncryption (1 2 840 113549 1 1 1)
   1119:             NULL
      :             }
   1121:           OCTET STRING
      :             41 3C 43 53 6B EC 3A C2 E4 E2 1B 69 80 1B 68 54
      :             80 81 7F 33 05 DD 67 E8 ED D0 03 A0 90 4B AA 43
      :             D4 54 CA 04 C9 78 97 8A E7 93 C0 05 F6 FA 30 BC
      :             59 1B 5D 30 5D E3 92 94 BA 4D D6 23 C0 59 17 F2
      :             0A F5 2C 73 0B 54 26 11 C3 3E FE 4C C2 ED 0B 89
      :             30 15 55 38 4A 80 D1 D5 AA 11 89 3A 9D 4B 47 C4
      :             29 F9 CF B7 44 53 21 E0 36 7E 81 02 CC DB 4C 09
      :             2D CA A1 AA 1B 76 F9 83 5C 86 53 24 30 BD 94 69
      :           }
      :         }
      :       }
      :     }
      :   }
```

6. Test Notes

This section describes some common interoperability problems. Implementers should verify that their clients do the correct things and perhaps make their clients forgiving in what they receive, or at least have them produce reasonable error messages when interacting with software that has these problems.

A common problem in interoperability is that some SIP clients do not support TLS and only do SSLv3. Check that the client does use TLS.

Many SIP clients were found to accept expired certificates with no warning or error.

TLS and S/MIME can provide the identity of the peer that a client is communicating with in the Subject Alternative Name in the certificate. The software must check that this name corresponds to the identity the server is trying to contact. If a client is trying to set up a TLS connection to good.example.com and it gets a TLS connection set up with a server that presents a valid certificate but with the name evil.example.com, it must generate an error or warning of some type. Similarly with S/MIME, if a user is trying to communicate with fluffy@example.com, the Subject Alternate Name field

in the certificate must match the AOR for fluffy.

Some implementations used binary MIME encodings while others used base64. There is no reason not to use binary - check that your implementation sends binary and preferably receives both.

7. Open Issues

The examples here attach the sender's certificates - is this how we want to go?

Need to add Accept header field value with multipart to all of the examples. Might also want to request congestion safety on all of them.

8. IANA Considerations

No IANA actions are required.

9. Acknowledgments

Many thanks to the developers of all the open source software used to create these call flows. This includes the underling crypto and TLS software used from openssl.org, the SIP stack from www.resiprocate.org, and the SIMPLE IMPP agent from www.sipimp.org. The TLS flow dumps were done with SSLDump from http://www.rtfm.com/ssldump. The book SSL and TLS [11] was a huge help in developing the code for these flows and is a great resource for anyone trying to implement TLS with SIP.

Thanks to Dan Wing and Robert Sparks for catching many silly mistakes and to Tat Chan who caught a key problem in what the signature was being computed over. Also thanks to Lyndsay Campbell.

10. References

10.1 Normative References

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

[2] 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.

[3] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[4] Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A., and P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[5] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

[6] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[7] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

[8] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3369, August 2002.

[9] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

10.2 Informative References

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

[11] Rescorla, E., "SSL and TLS - Designing and Building Secure Systems", 2001.

Authors' Addresses

Cullen Jennings
Cisco Systems
170 West Tasman Drive
Mailstop SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 902-3341
Email: fluffy@cisco.com

Kumiko Ono
NTT Corporation
Musashino-shi, Tokyo 180-8585
Japan

Phone: +81 422 59 4508
Email: ono.kumiko@lab.ntt.co.jp

Appendix A. Making Test Certificates

These scripts allow you to make certificates for test purposes. The certificates will all share a common CA root so that everyone running these scripts can have interoperable certificates. WARNING - these certificates are totally insecure and are for test purposes only. All the CA created by this script share the same private key to facilitate interoperability testing, but this totally breaks the security since the private key of the CA is well known.

The instructions assume a Unix-like environment with openssl installed, but openssl does work in Windows too. Make sure you have openssl installed by trying to run "openssl". Run the makeCA script found in Appendix A.1; this creates a subdirectory called demoCA. If the makeCA script cannot find where your openssl is installed you will have to set an environment variable called OPENSSLDIR to whatever directory contains the file openssl.cnf. You can find this with a "locate openssl.cnf". You are now ready to make certificates.

To create certs for use with TLS, run the makeCert script found in Appendix A.2 with the fully qualified domain name of the proxy you are making the certificate for. For example, "makeCert host.example.net". This will generate a private key and a certificate. The private key will be left in a file named domain_key_example.net.pem in pem format. The certificate will be in domain_cert_example.net.pem. Some programs expect both the certificate and private key combined together in a PKCS12 format file. This is created by the script and left in a file named example.net.p12. Some programs expect this file to have a .pfx extension instead of .p12 - just rename the file if needed. A filed with a certificate signing request, called example.net.csr, is also created an can be used to get the certificate signed by another CA.

A second argument indicating the number of days for which the certificate should be valid can be passed to the makeCert script. It is possible to make an expired certificate using the command "makeCert host.example.net 0".

Anywhere that a password is used to protect a certificate, the password is set to the string "password".

The root certificate for the CA is in the file root_cert_fluffyCA.pem.

For things that need DER format certificates, a certificate can be converted from PEM to DER with "openssl x509 -in cert.pem -inform PEM -out cert.der -outform DER".

Some programs expect certificates in PKCS#7 format (with a file extension of .p7c). You can convert these from PEM format with to PKCS#7 with "openssl crl2pkcs7 -nocrl -certfile cert.pem -certfile demoCA/cacert.pem -outform DER -out cert.p7c"

IE, Outlook, and Netscape can import and export .p12 files and .p7c files. You can convert a pkcs7 certificate to PEM format with "openssl pkcs7 -in cert.p7c -inform DER -outform PEM -out cert.pem".

The private key can be converted to pkcs8 format with "openssl pkcs8 -in a_key.pem -topk8 -outform DER -out a_key.p8c"

In general, a TLS client will just need the root certificate of the CA. A TLS server will need its private key and its certificate. These could be in two PEM files or one .p12 file. An S/MIME program will need its private key and certificate, the root certificate of the CA, and the certificate for every other user it communicates with.

A.1 makeCA script

```
#!/bin/sh
#set -x

rm -rf demoCA

mkdir demoCA
mkdir demoCA/certs
mkdir demoCA/crl
mkdir demoCA/newcerts
mkdir demoCA/private
#echo "01" > demoCA/serial
hexdump -n 4 -e '4/1 "%04d"' /dev/random > demoCA/serial
touch demoCA/index.txt

# You may need to modify this for where your default file is
# you can find where yours in by typing "openssl ca"
for D in /etc/ssl /usr/local/ssl /sw/etc/ssl /sw/share/ssl; do
        CONF=${OPENSSLDIR:=$D}/openssl.cnf
        [ -f ${CONF} ] && break
```

```
done

if [ ! -f $CONF  ]; then
    echo "Can not find file $CONF - set your OPENSSLDIR variable"
    exit
fi
cp $CONF openssl.cnf

cat >> openssl.cnf  <<EOF
[ cj_cert ]
subjectAltName=\${ENV::ALTNAME}
basicConstraints=CA:FALSE
subjectKeyIdentifier=hash
#authorityKeyIdentifier=keyid,issuer:always

[ cj_req ]
basicConstraints = CA:FALSE
subjectAltName=\${ENV::ALTNAME}
subjectKeyIdentifier=hash
#authorityKeyIdentifier=keyid,issuer:always
#keyUsage = nonRepudiation, digitalSignature, keyEncipherment

EOF

cat > demoCA/private/cakey.pem <<EOF
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,4B47A0A73ADE342E

aHmlPa+ZrOV6v+Jk0SClxzpxoG3j0ZuyoVkF9rzq2bZkzVBKLU6xhWwjMDqwA8dH
3fCRLhMGIUVnmymXYhTW9svI1gpFxMBQHJcKpV/SmgFn/fbYk98Smo2izHOniIiu
NOu2zr+bMiaBphOAZ/OCtVUxUOoBDKN9lR39UCDOgkEQzp9Vbw7l736yu5H9GMHP
JtGLJyx3RhS3TvLfLAJZhjm/wZ/9QM8GjyJEiDhMQRJVeIZGvv4Yr1u6yYHiHfjX
tX2eds8Luc83HbSvjAyjnkLtJsAZ/8cFzrd7pjFzbogLdWuil+kpkkf5h1uzh7oa
um0M1EXBE4tcDHsfg1iqEsDMIei/U+/rWfk1PrzYlklwZp8S03vulkDm1fT76W7d
mRBg4+CrHA6qYn6EPWB37OBtfEqAfINnIcI1dWzso9A0bTPD4EJO0JA0PcZ/2JgT
PaKySgooHQ8AHNQebelch6M5LFExpaOADJKrqauKcc2HeUxXaYIpac5/7drIl3io
UloqUnMlGa3eLP7BZIMsZKCfHZ8oqwU4g6mmmJath2gODRDx3mfhH6yaimDL7v4i
SAIIkrEHXfSyovrTJymfSfQtYxUraVZDqax6oj/eGllRxliGfMLYG9ceU+yU/8FN
LE7P+Cs19H5tHHzx1LlieaK43u/XvbXHlB5mqL/fZdkUIBJsjbBVx0HR8eQl2CH9
YJDMOPLADecwHoyKA0AY59oN9d41oF7yZtN9KwNdslROYH7mNJlqMMenhXCLN+Nz
vVU5/7/ugZFhZqfS46c1WdmSvuqpDp7TBtMeaH/PXjysBr0iZffOxQ==
-----END RSA PRIVATE KEY-----
EOF

cat > demoCA/cacert.pem <<EOF
-----BEGIN CERTIFICATE-----
MIIDJDCCAo2gAwIBAgIBADANBgkqhkiG9w0BAQUFADBwMQswCQYDVQQGEwJVUzET
```

```
MBEGA1UECBMKQ2FsaWZvcm5pYTERMA8GA1UEBxMIU2FuIEpvc2UxDjAMBgNVBAoT
BXNpcGl0MSkwJwYDVQQLEyBTaXBpdCBUZXN0IENlcnRpZmljYXRlIEF1dGhvcml0
eTAeFw0wMzA3MTgxMjIxNTJaFw0xMzA3MTUxMjIxNTJaMHAxCzAJBgNVBAYTAlVT
MRMwEQYDVQQIEwpDYWxpZm9ybmlhMREwDwYDVQQHEwhTYW4gSm9zZTEOMAwGA1UE
ChMFc2lwaXQxKTAnBgNVBAsTIFNpcGl0IFRlc3QgQ2VydGlmaWNhdGUgQXV0aG9y
aXR5MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDIh6DkcUDLDyK9BEUxkud
+nJ4xrCVGKfgjHm6XaSuHiEtnfELHM+9WymzkBNzZpJu30yzsxwfKoIKugdNUrD4
N3viCicwcN35LgP/KnbN34cavXHr4ZlqxH+OdKB3hQTpQa38A7YXdaoz6goW2ft5
Mi74z03GNKP/G9BoKOGd5QIDAQABo4HNMIHKMB0GA1UdDgQWBBRrRhcU6pR2JYBU
bhNU2qHjVBShtjCBmgYDVR0jBIGSMIGPgBRrRhcU6pR2JYBUbhNU2qHjVBShtqF0
pHIwcDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExETAPBgNVBAcT
CFNhbiBKb3NlMQ4wDAYDVQQKEwVzaXBpdDEpMCcGA1UECxMgU2lwaXQgVGVzdCBD
ZXJ0aWZpY2F0ZSBBdXRob3JpdHmCAQAwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0B
AQUFAAOBgQCWbRvv1ZGTRXxbH8/EqkdSCzSoUPrs+rQqR0xdQac9wNY/nlZbkR3O
qAezG6Sfmklvf+DOg5RxQq/+Y6I03LRepc7KeVDpaplMFGnpfKsibETMipwzayNQ
QgUf4cKBiF+65Ue7hZuDJa2EMv8qW4twEhGDYclpFU9YozyS1OhvUg==
-----END CERTIFICATE-----
EOF


# uncomment the following lines to generate your own key pair

#openssl req -newkey rsa:1024 -passin pass:password \
#    -passout pass:password \
#    -sha1 -x509 -keyout demoCA/private/cakey.pem \
#    -out demoCA/cacert.pem -days 3650 <<EOF
#US
#California
#San Jose
#sipit
#Sipit Test Certificate Authority
#
#
#EOF

openssl crl2pkcs7 -nocrl -certfile demoCA/cacert.pem \
        -outform DER -out demoCA/cacert.p7c

cp demoCA/cacert.pem root_cert_fluffyCA.pem
```

A.2 makeCert script

```
#!/bin/sh
#set -x

if [  $# == 1  ]; then
```

```
    DAYS=1095
  elif [ $# == 2 ]; then
    DAYS=$2
  else
    echo "Usage: makeCert test.example.org [days]"
    echo "       makeCert alice@example.org [days]"
    echo "days is how long the certificate is valid"
    echo "days set to 0 generates an invalid certificate"
    exit 0
  fi

  ADDR=$1

  echo "making cert for ${ADDR}"

  rm -f ${ADDR}_*.pem
  rm -f ${ADDR}.p12

  case ${ADDR} in
  *:*) ALTNAME="URI:${ADDR}" ;;
  *@*) ALTNAME="URI:sip:${ADDR},URI:im:${ADDR},URI:pres:${ADDR}" ;;
  *)   ALTNAME="DNS:${ADDR}" ;;
  esac

  rm -f demoCA/index.txt
  touch demoCA/index.txt
  rm -f demoCA/newcerts/*

  export ALTNAME

  openssl genrsa  -out ${ADDR}_key.pem 1024
  openssl req -new  -config openssl.cnf -reqexts cj_req \
          -sha1 -key ${ADDR}_key.pem \
          -out ${ADDR}.csr -days ${DAYS} <<EOF
  US
  California
  San Jose
  sipit

  ${ADDR}



  EOF

  if [ $DAYS == 0 ]; then
  openssl ca -extensions cj_cert -config openssl.cnf \
      -passin pass:password -policy policy_anything \
```

```
      -md sha1 -batch -notext -out ${ADDR}_cert.pem \
      -startdate 9901010000000Z \
      -enddate 000101000000Z \
       -infiles ${ADDR}.csr
  else
  openssl ca -extensions cj_cert -config openssl.cnf \
      -passin pass:password -policy policy_anything \
      -md sha1 -days ${DAYS} -batch -notext -out ${ADDR}_cert.pem \
       -infiles ${ADDR}.csr
  fi

  openssl pkcs12 -passin pass:password \
      -passout pass:password -export \
      -out ${ADDR}.p12 -in ${ADDR}_cert.pem \
      -inkey ${ADDR}_key.pem -name ${ADDR} -certfile demoCA/cacert.pem

  openssl x509 -in ${ADDR}_cert.pem -noout -text

  case ${ADDR} in
  *@*) mv ${ADDR}_key.pem user_key_${ADDR}.pem;   \
       mv ${ADDR}_cert.pem user_cert_${ADDR}.pem ;;
  *)   mv ${ADDR}_key.pem domain_key_${ADDR}.pem; \
       mv ${ADDR}_cert.pem domain_cert_${ADDR}.pem ;;
  esac
```

Appendix B. Certificates for Testing

This section contains various certificates used for testing in PEM format.

Fluffy's certificate.

```
-----BEGIN CERTIFICATE-----
MIICzjCCAjegAwIBAgIIAZUAcQIzAFgwDQYJKoZIhvcNAQEFBQAwcDELMAkGA1UE
BhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExETAPBgNVBAcTCFNhbiBKb3NlMQ4w
DAYDVQQKEwVzaXBpdDEpMCcGA1UECxMgU2lwaXQgVGVzdCBDZXJ0aWZpY2F0ZSBB
dXRob3JpdHkwHhcNMDUwMjAzMTg0OTM0WhcNMDgwMjAzMTg0OTM0WjBiMQswCQYD
VQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTERMA8GA1UEBxMIU2FuIEpvc2Ux
DjAMBgNVBAoTBXNpcGl0MRswGQYDVQQDFBJmbHVmZnlAZXhhbXBsZS5jb20wgZ8w
DQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMqrm5tOPNVFPM4ApjaouezSduK5m+go
qrqGIsXPMz5PbVYhrr1UhHwUFPl9mYUATpPW/WvU0dRVjsmJsa8rXyOZSpXlaGVk
HRKn29PVlxhHNZzmiCedqGzKKoTmYtjx6aIaOX4OD5ClpnkhvCpntN1pkIKarh8C
UopY0/XQ1GZnAgMBAAGjfzB9MFEGA1UdEQRKMEiGFnNpcDpmbHVmZnlAZXhhbXBs
ZS5jb22GFWltOmZsdWZmeUBleGFtcGxlLmNvbYYXcHJlczpmbHVmZnlAZXhhbXBs
ZS5jb20wCQYDVR0TBAIwADAdBgNVHQ4EFgQU7NqYXun399fsKylL2iXux8d+lXAw
DQYJKoZIhvcNAQEFBQADgYEATEZJbgFI4tRu10ih83vIpZg3pURGWJ9KN32Q+1//
Nr1nMfAp3gri6rnwXJ+toN7lTkKPEdhB6mi+28Ie+uWKLX9mEynp2o/7gL9+XrYE
rQheWJW3xTiF1WUxrYDLKKdMrRH9QTs3d1rehZY9Zutfmvhgx46x/EpDU7YRTS70
Pf8=
-----END CERTIFICATE-----
```

Fluffy’s private key

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDKq5ubTjzVRTzOAKY2qLns0nbiuZvoKKq6hiLFzzM+T21WIa69
VIR8FBT5fZmFAE6T1v1r1NHUVY7JibGvK18jmUqV5WhlZB0Sp9vT1ZcYRzWc5ogn
nahsyiqE5mLY8emiGjl+Dg+QpaZ5IbwqZ7TdaZCCmq4fAlKKWNP10NRmZwIDAQAB
AoGAXgtxwoh0jBZ716/PcS+sTut+xUiRwxIT30fdHONACRr8RmqM1khAzf7XmMoi
kegJjmrF3+K6l4g4IOcnL3y1wVCtzJ1f2QDTuVzAsvazZqI4+pNB4LaAb+JPNQ+4
BtrQSXADXv7HfkUakzeZpgnJYw+zHWaVogKjcLDKHWdrbOECQQDpH/G+GsJ4mnrp
wZF9OxKqKhqBO73ZONHDxu55AukLghGnFh1udqdCQ7EPsaCqLN82RS4gn/WDfnBh
WB8DRavxAkEA3o6nMOMyKdsuqBbGyEPvaPDVmw973wtEohIj6MgwdYSUOhdKAurR
hs09yVGy0QpjoNHIE0vi5lUhPxJ1+Xvv1wJBAL0Ry14DFfX6U/WBqB2I63pW62gk
q7ShAH9nt8EtOxS6SNbaeMQ+Nyjm/ZNc3JEoE2BQezi6gsRCp6JLlduRhgECQD1p
V7EhwCHUnVc8kbWJKXLnocmbyC6PyWx/XPFK7DRBVTWCX6XWbeKol7gJlzIfj8Y8
nNzWP9IXA4mH6o3hKRkCQA+1er++Tx24uypEijIi7OK0bfjJUlrhCM9NVWxDKrzO
3zpuUB7yzuxrbcMZI8JKQIHL0sWz7egscepxS+N61y8=
-----END RSA PRIVATE KEY-----
```

Kumiko’s certificate

```
-----BEGIN CERTIFICATE-----
MIICzjCCAjegAwIBAgIIAZUAcQIzAFcwDQYJKoZIhvcNAQEFBQAwcDELMAkGA1UE
BhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExETAPBgNVBAcTCFNhbiBKb3NlMQ4w
DAYDVQQKEwVzaXBpdDEpMCcGA1UECxMgU2lwaXQgVGVzdCBDZXJ0aWZpY2F0ZSBB
dXRob3JpdHkwHhcNMDUwMjAzMTg0OTIzWhcNMDgwMjAzMTg0OTIzWjBiMQswCQYD
VQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTERMA8GA1UEBxMIU2FuIEpvc2Ux
DjAMBgNVBAoTBXNpcGl0MRswGQYDVQQDFBJrdW1pa29AZXhhbXBsZS5uZXQwgZ8w
DQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANX6dhOhUuf+2I3lymzeuSDwHLZqMqnu
3ISiIji/VlEoVUBFIHYjtxbmhIi40mEl4cqT+tVI6gY6Pe7VrL835Yr3AoLLeUB7
4mXa7T152+jAxA4+nCVnIAkMrPxTDeBFEfn+qyCRPWyQ7WEgH3Vd9AufnC7aeafD
pp+dcAOFZ2pBAgMBAAGjfzB9MFEGA1UdEQRKMEiGFnNpcDprdW1pa29AZXhhbXBs
ZS5uZXSGFWltOmt1bWlrb0BleGFtcGxlLm5ldIYXcHJlczprdW1pa29AZXhhbXBs
ZS5uZXQwCQYDVR0TBAIwADAdBgNVHQ4EFgQUNi5qQQ2G6AsiZK79cPEXYmPsqFIw
DQYJKoZIhvcNAQEFBQADgYEABIFd9N/3/05AD7Kt9kKSdy6vFvncU1IaccuFfdXc
QPfewY8NWwYKWsu588D4Nu77VQ++6a8AtjlJPSY/742Z4oKq1jfxdA+Uz/Z9cv2v
6aM4oX7R5FTgJTbHRC0ueH32OhNlcLhSNGHzNWSrS8AbtNOlfLRJipZI3N0W5b6q
09Q=
-----END CERTIFICATE-----
```

Kumiko's private key

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDV+nYToVLn/tiN5cps3rkg8By2ajKp7tyEoiI4v1ZRKFVARSB2
I7cW5oSIuNJhJeHKk/rVSOoGOj3u1ay/N+WK9wKCy3lAe+Jl2u09edvowMQOPpwl
ZyAJDKz8Uw3gRRH5/qsgkT1skO1hIB91XfQLn5wu2nmnw6afnXADhWdqQQIDAQAB
AoGBANJktWrxyanxC47iLdpEWHVJgoHeA7jQ8yS6orl3cPDVnpVWIufmkCTFPfWM
/Namv89HF3BVhD3hUHogwP03gcsIdxpccnu1wnmTW7IhSQXjBts0mEDbOw8S+WtS
9NjRI4m1+86OflE+TVa3DtwCE/pEOKhFvcZHvXiosYMnucABAkEA6xqKEwR1zI/V
u2B28Lcv0iafkJQDfPB3ooahQ+9qy5qUWgGZzXj6tM8YUusVqR/NCg8auqRC5uWD
yonN98phQQJBAOj/Pp9yyO2NCVs4Mp5QSXDOlRAOuruMz6vlmURQO/8uBmHvETfC
nkvqxxHjHW7mmusEY+ZIvRxmFV4RZcYByQECQHiT5/TQ+Mmti2TKmLXkffY+MOAp
yZAulG0at2LsS82YvjVbVNJ5Fbvd6w+72iQfVz2teXv3+wgI9orOGoDXnwECQGrE
I58PCzGHkkUBkHhpE+4kS7wK89hjYvpDAKOEHKoHHhecZAhoHv9suwHgT6l09IJD
BcANjtLHmHz9feRpBwECQQCuIn02CMxFy5yhjj4nlmCRQ6w6KBWjY68xnN4Qj/g3
SV+1HtmCclS0bK7e/IV6gOKn+MV3C+14JGdSRM+9HqcZ
-----END RSA PRIVATE KEY-----
```

Certificate for example.com

```
-----BEGIN CERTIFICATE-----
MIICjDCCAfWgAwIBAgIIAZUAcQIzAFUwDQYJKoZIhvcNAQEFBQAwcDELMAkGA1UE
BhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExETAPBgNVBAcTCFNhbiBKb3NlMQ4w
DAYDVQQKEwVzaXBpdDEpMCcGA1UECxMgU2lwaXQgVGVzdCBDZXJ0aWZpY2F0ZSBB
dXRob3JpdHkwHhcNMDUwMjAzMTg0OTA4WhcNMDgwMjAzMTg0OTA4WjBbMQswCQYD
VQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTERMA8GA1UEBxMIU2FuIEpvc2Ux
DjAMBgNVBAoTBXNpcGl0MRQwEgYDVQQDEwtleGFtcGxlLmNvbTCBnzANBgkqhkiG
9w0BAQEFAAOBjQAwgYkCgYEA5jF2tSfMjTKFVnD3wjMzMiZCXjxocXsfeVDQcic7
Sq/yztEMvMBfMWpD53ytZL3H5iWfqs0tkKpohGJ7Bb5Dpa+76p2pW6RTnSKL2pYu
Hz+SRrjMyCQ8Rs1dLWSFsaTKAfGOxX4P/wCRo+rLPhICdaS7CMjQKu+zu3J6mOX/
n4ECAwEAAaNEMEIwFgYDVR0RBA8wDYILZXhhbXBsZS5jb20wCQYDVR0TBAIwADAd
BgNVHQ4EFgQUIurLOGYd8ZYMmke2uxxSRLB3ZY0wDQYJKoZIhvcNAQEFBQADgYEA
rutJ7R7xjSapbQOCktXfRMQeHwd1iDfkdpc1ElmYeXgWbjuxwCvbhQJrdMlbGZLa
fvVBC7zS3UWqb74k3EhXZtkugt+ejXADc3Xvj3pWTMxCvTFFsF7/0TvEgu79p8EQ
NOuBSRprhn7HYR2zuQoCvYT4R6/P8ahzqDEdIHoGf6w=
-----END CERTIFICATE-----
```

Private key for example.com

```
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDmMXa1J8yNMoVWcPfCMzMyJkJePGhxex95UNByJztKr/LO0Qy8
wF8xakPnfK1kvcfmJZ+qzS2QqmiEYnsFvkOlr7vqnalbpFOdIovali4fP5JGuMzI
JDxGzV0tZIWxpMoB8Y7Ffg//AJGj6ss+EgJ1pLsIyNAq77O7cnqY5f+fgQIDAQAB
AoGBANtRm2FkRv7seJ/wSA6OS6PnUeqJMZWVklo6xi9M86/oTbYA9VrNCqWBMqtW
XboTG2dKx4KrtFMWGTiwv7esHLPsUB1jYF7/KEsRh4WoRxfeWoQlAY6VYXycg6b5
X0uORdFMWL+WRxPmo8IhDKEwNyRyCyGQjfKpMj0724WjEqWxAkEA9MFDUQD+fL3N
ImRQl9ns3nHIIbcrtfxGCFaj+EJEwsyc5gq7QxRc3niNVt5pogPP7+CxskLaPPKU
TJmhtwixLQJBAPDE7hcDCPtsn9DIOXf/ZxXjfZAlAfwVsT+ggWQi5r63lGwjIbCT
qO6TijtbSqqD0QqULTabVwpIdYyknQqQlCUCQGnkG322UmQhsdiJUh0Amex7ibyc
hPrNVHdTFMnZ0en9oHwedHpHGw7dVTkaLNV9lL8RlY+sQMNRqDuj1EVeK1kCQQCH
945FLI+b/OHbs9bQb0k10TyNdHjEdTOdrPSlKhiIx39n+gcCgsC5ylQb5RgrZzlb
8gX+eocS5YyMmkGdP7yJAkEAsmGKAgt4nTfZY5L8PytPK8lCJjBLcyIllI3QEiMY
K/81YWYQcqsg5/cLBZC26KgNvxkyLwxS220Djlm19HJKGQ==
-----END RSA PRIVATE KEY-----
```

Certificate for example.net

```
-----BEGIN CERTIFICATE-----
MIICjDCCAfWgAwIBAgIIAZUAcQIzAFYwDQYJKoZIhvcNAQEFBQAwcDELMAkGA1UE
BhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExETAPBgNVBAcTCFNhbiBKb3NlMQ4w
DAYDVQQKEwVzaXBpdDEpMCcGA1UECxMgU2lwaXQgVGVzdCBDZXJ0aWZpY2F0ZSBB
dXRob3JpdHkwHhcNMDUwMjAzMTg0OTExWhcNMDgwMjAzMTg0OTExWjBbMQswCQYD
VQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTERMA8GA1UEBxMIU2FuIEpvc2Ux
DjAMBgNVBAoTBXNpcGl0MRQwEgYDVQQDEwtleGFtcGxlLm5ldDCBnzANBgkqhkiG
9w0BAQEFAAOBjQAwgYkCgYEA2w4I/bz/vxzVskUEF56EYjf4yUftpG8jhmIiwsA8
AKLwc7CTnceW+tLmdDfUQLWw+HP4ky0tgQQA6pmviPORUNjuSj91dE7EJk3ZKePE
3MZ2M5JL6CEFn3HEFnHOQKv3TMKIGSpUZJjHmm15yRPiAlx0Q2vJ29h4W52X1DPM
62MCAwEAAaNEMEIwFgYDVR0RBA8wDYILZXhhbXBsZS5uZXQwCQYDVR0TBAIwADAd
BgNVHQ4EFgQUHNoIc7Or6o1iTsM1PmWPdgbxUAwwDQYJKoZIhvcNAQEFBQADgYEA
VlSod7+XfvSKNsybqtWPaM8VnoRLFVXvukgQbsdv4wuv5bnDfwxdU25rdizBbql+
m8Us+ky8ORw190v73mSeOro7KMv0mN1u2BaGUB/wjaRsH2HC+UZb0ok3vzZ+W8Re
ECjcVyHNRGVw5Iu2W5iWcO/a/74vPaVBiFQQJBRSLxg=
-----END CERTIFICATE-----<
```

Private key for example.net

```
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDbDgj9vP+/HNWyRQQXnoRiN/jJR+2kbyOGYiLCwDwAovBzsJOd
x5b60uZ0N9RAtbD4c/iTLS2BBADqma+I85FQ2O5KP3V0TsQmTdkp48TcxnYzkkvo
IQWfccQWcc5Aq/dMwogZKlRkmMeabXnJE+ICXHRDa8nb2HhbnZfUM8zrYwIDAQAB
AoGBAIrUP1CIutEldi3wXaKWfTI+ZPc0FeFz6mDdy0gAS0bf/WJk03lYqFA434Ni
aqvEOu+LmEu2gzNUFTyZwE0ciMg3NQ0H57z7OvbnHa0LajiJROo7zkROrmE5GTIV
v2WstOKJYsMdcTVa4VZd9cHH6zWXHtWDT+Y2MxrIerFnOYxBAkEA72cBQSE4SStZ
KvodDuMjFXG97Z1F927Xe/47iWnYRKhVB/jwN9uYpJog2cQFgsIsRMltozi3huTP
L8IKkI5N4QJBAOo95ShiRPcbXIXY1IcUGx1Rulr+paIAJwjuuutwrtCA1CbIKB0j
vfGVr3mKBGV2XLmz15nNV+5WFiLRBiUgucMCQQCxf+63KnlADurS6ZTH5/KoQKfw
WE568WzFWy8raBXYefJpsdHxqFiZmklHDIaFd5A5BBvNDA1O77EKGNWablghAkEA
zbvpPqv4+LRuchy8pZtyKTE0JWHNZlkN79mGEO4ajITqUNmx6c4PsVUQFwayz87C
qFQdxDdHyMyRiqjd5dQ1cwJAfJsXNGcOhilkV3xBy95tb3IsVP6G5DqwtID4hrYa
Onf9xrVzh9M29Xp+AHcwS4Y0+UgiNrd5BlbZs+ALZPD/jw==
-----END RSA PRIVATE KEY-----
```

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

WG
Internet-Draft
Expires: January 16, 2006

R. Mahy
SIP Edge LLC
July 15, 2005

A Location Event Package using the Session Initiation Protocol (SIP)

draft-mahy-geopriv-sip-loc-pkg-01.txt

Status of this Memo

Copyright Notice

Abstract

This document describes a Session Initiation Protocol (SIP) event package to carry location data about named SIP resources. Inherent in this event package are filters which limit notifications to compelling events which are described by the filters. The resulting location information is conveyed in existing location formats wrapped in GEOPRIV privacy extensions to the Presence Information Document Format (PIDF-LO). Location disclosure is limited to voluntary disclosure by a notifier that possesses credentials for the named resource.

Table of Contents

1. 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 RFC-2119 [4].

2. Overview

Conveying PIDF-LO [3] bodies in most SIP [1] messages is straightforward protocol usage defined in [15]. In addition, a SIP event [2] package for location is an obvious usage of existing SIP capabilities. However the difficult part about asynchronous notification of location information is that many forms of location are measured as a continous gradient. Unlike notications using discreet quanties, it is difficult to know when a change in location is large enough to warrant notifications. Moreover, different applications require a wide variety of location resolutions. Any optimization made for one application would ultimately result in wasteful polling or a sluggish user interface for other applications.

The mechanism described here defines filters in XML [5] documents which limit location notification to events which are of relevance to the subscriber. These filters are provided in the body of SIP subscription requests and persist for the duration of the subscription or until they are changed in an updated SIP subscription request with a replacement filter.

In addition to the relevant filters, this document also defines a new XML schema [6] which can be included in PIDF-LO documents to indicate that the resource is inside or outside of a container region.

3. Definition of Location Filter Format

The granularity of notifications necessary for various geographic location applications varies dramatically. The subscriber should be able to get asynchronous notifications with appropriate granularity and accuracy, without having to poll or flood the network with notifications which are not important to the application. Notifications should only happen when the notification would be considered an Interesting Event to the subscriber. Subscriptions to this event package contain a filter document in the XML document format defined in this section. The terminal elements in this format are defined in terms of existing Geographic Markup Language (GML) [10] data types.

The notifications are in PIDF-LO (by default) or any other format acceptable to both the subscriber and notifier. The selection of a subset of GML or specific location format capabilities contained in a PIDF-LO body is a generic issue for the GEOPRIV Working Group

to define, and is out of the scope of this document.

This document defines the following as an initial list of Interesting Events:

1. the resource moves more than a specific distance horizontally or vertically since the last notification
2. the resource exceeds a specific speed
3. the resource enters or exits one or more GML objects (for example, a set of 2-dimensional or 3-dimensional regions) included or referenced in the filter.
4. one or more of the values of the specified address labels has changed for the resource (for example, the A1 value of the civilAddress has changed from California to Nevada)

This specification makes use of XML namespaces [7] for identifying location filter documents and document fragments. The namespace URI for elements defined by this specification is a URN [11], using the namespace identifier 'ietf' defined by [12] and extended by [13]. This URN is:

urn:ietf:params:xml:ns:location-filter

The filter format starts with a top-level XML element called "<location-filter>", which contains one or more filter events. The semantics of multiple elements inside a location-filter is a logical OR. In other words, if any of the individual filter events occurs, the event satisfies the location-filter and triggers a notification.

The movedHoriz and movedVert filter events each indicate a minimum horizontal motion or vertical distance (respectively) that the resource must have moved from the location of the resource when the last notification was sent in order to trigger this event. The distance is measured absolutely from the point of last notification rather than in terms of cumulative motion (For example, someone pacing inside a room will not trigger an event if the trigger threshold is slightly larger than the room.) Each of these events can only appear once in a location-filter. These events have an attribute "uom" (for "units of measure"), which indicates the units of the element. The default unit for these events is meters.

Similarly, the speedExceeds filter event indicates a minimum horizontal speed of the resource before the speedExceeds event is triggered. This element can appear only once in a location-filter, and has a "uom" attribute which defaults to meters per second if not present.

This filter measures the horizontal component of speed in any direction. It does not measure velocity. Note also that there is no corresponding event triggered when speed drops below a threshold.

Below are some examples. In the first example if the resource moves 20m in the x,y direction or 3m in the z direction, send a notification:

```
<location-filter>
  <movedHoriz uom="#meters">20</movedHoriz>
  <movedVert uom="#meters">3</movedVert>
</location-filter>
```

If the resource exceeds 3 meters per second (10.8 km/h), send a notification:

```
<location-filter>
  <speedExceeds uom="#mps">3</speedExceeds>
</location-filter>
```

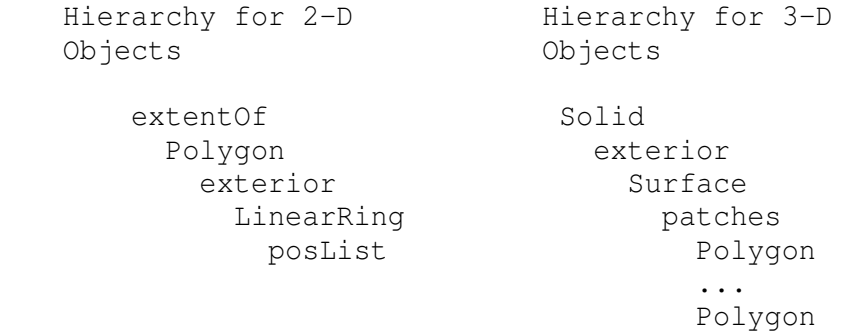
The valueChanges filter event contains a string which is interpreted as an XPath [8] expression evaluated within the context of the location-info element of the PIDF-LO document which would be generated by the notification. The XPath expression MUST evaluate to only a single Xpath node. If the value of any of the elements in the resulting node changes, then the filter event is triggered. Note that the value of the resulting node changes if any of those nodes or subnodes transitions from having a value to having no value or vice versa. A location-filter may contain multiple valueChanges filters.

For example, given the following logical PIDF-LO document, If the state (A1), county (A2), city (A3), or postal code (PC) changes, send a notification:

```
PIDF-LO Location Document:
    <?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
       xmlns:gp="urn:ietf:params:xml:ns:pidf:geopriv10"
       xmlns:cl="urn:ietf:params:xml:ns:pidf:geopriv10:civilLoc"
       entity="pres:geotarget@example.com">
     <tuple id="sg89ae">
      <status>
       <gp:geopriv>
         <gp:location-info>
           <cl:civilAddress>
             <cl:country>US</cl:country>
             <cl:A1>New York</cl:A1>
             <cl:A3>New York</cl:A3>
             <cl:A6>Broadway</cl:A6>
             <cl:HNO>123</cl:HNO>
             <cl:LOC>Suite 75</cl:LOC>
             <cl:PC>10027</cl:PC>
           </cl:civilAddress>
         </gp:location-info>
         <gp:usage-rules>
           <gp:retransmission-allowed>yes</gp:retransmission-allowed>
           <gp:retention-expiry>2003-06-23T04:57:29Z
           </gp:retention-expiry>
         </gp:usage-rules>
       </gp:geopriv>
      </status>
      <timestamp>2003-06-22T20:57:29Z</timestamp>
     </tuple>
    </presence>

Filter Document:
  <location-filter
    xmlns="urn:ietf:params:xml:ns:location-filter"
    xmlns:cl="urn:ietf:params:xml:ns:pidf:geopriv10:civilLoc">
    <valueChanges>cl:civilAddress/cl:A1</valueChanges>
    <valueChanges>cl:civilAddress/cl:A2</valueChanges>
    <valueChanges>cl:civilAddress/cl:A3</valueChanges>
    <valueChanges>cl:civilAddress/cl:PC</valueChanges>
  </location-filter>
```

Finally, the "enterOrExit" filter event is triggered when the resource enters or exits a named 2-dimensional or 3-dimensional region or list of regions corresponding to a GML feature. These regions can be defined using inline snippets of GML, or externally referenced using a URI (Uniform Resource Identifier). Notifiers which support this document MUST be able to support 2-dimensional regions and lists of regions, for which the regions can be defined in

terms of the GML extentOf a Polygon defined using an exterior LinearRing object. These Polygons are defined using the hierarchy in the figure below.



Similarly, Notifiers MUST be able to support 3-dimensional regions which can be defined as a fixed height vertical projection of such a 2-dimensional Polygon, and lists thereof. Specifically, these are GML Solids defined in terms of an exterior Surface of polygonal patches, such that all included Polygons are either parallel (horizontal) or perpedicular (vertical) to the geoid.

The posList for any 2-dimensional region MUST be defined using the EPSG 4326 coordinate reference system. The posList for any 3-dimensional region MUST be defined using the EPSG 4979 coordinate reference system. A location-filter can contain more than one enterOrExit filter event.

Notifiers MAY support other more complex geometries or additional coordinate reference systems. How the Subscriber negotiates support for more complex geometries or reference systems is out of the scope of this document.

Likewise, this document does not describe how a subscriber discovers the existence of externally referenced features. This topic is out of scope of this document.

In most cases Subscribers that use location filters based on enterOrExit events are especially interested in the resource's relationship to those named features. Consequently, the notifier MUST include either a "containment" element for each feature mentioned in the location-filter which has changed its containment properties with respect to the resource since the last notification. These elements are defined in Section 4. The notifier MAY include any other form of location that is relevant.

For example, if the resource enters or exits Building 10 (which is defined by specific 2-D or 3-D rectangular coordinates), send a notification:

```
   Version in 2-Dimensions:
   <location-filter>
     <enterOrExit>
       <my:Building>
         <gml:name>Building 10</gml:name>
         <gml:extentOf>
           <gml:Polygon>
             <gml:exterior>
               <gml:LinearRing>
                 <gml:posList
        srsName="http://www.opengis.net/gml/srs/epsg.xml/#4979">
                      37.41188 -121.93243 0
                      37.41188 -121.93132 0
                      37.41142 -121.93132 0
                      37.41142 -121.93242 0
                      37.41188 -121.93243 0
                 </gml:posLis>
               </gml:LinearRing>
             </gml:exterior>
           </gml:Polygon>
         </gml:extentOf>
       </my:Building>
     </enterOrExit>
   </location-filter>

   Version in 3-Dimensions:
   <location-filter>
     <enterOrExit>
       <my:Building>
         <gml:name>Building 10</gml:name>
         <gml:Solid>
           <gml:exterior>
             <gml:Surface>
               <gml:patches>
                 <gml:Polygon> <!-- floor -->
                   <gml:exterior>
                     <gml:LinearRing>
                       <gml:posList
        srsName="http://www.opengis.net/gml/srs/epsg.xml/#4979">
                           37.41188 -121.93243 0
                           37.41188 -121.93132 0
                           37.41142 -121.93132 0
                           37.41142 -121.93242 0
                           37.41188 -121.93243 0
                       </gml:posLis>
                     </gml:LinearRing>
                   </gml:exterior>
                 </gml:Polygon>
```

```
            <gml:Polygon> <!-- north wall -->
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList
      srsName="http://www.opengis.net/gml/srs/epsg.xml/#4979">
                       37.41188 -121.93243 0
                       37.41188 -121.93243 0
                       37.41188 -121.93132 25
                       37.41188 -121.93132 25
                       37.41188 -121.93243 0
                  </gml:posLis>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
            <gml:Polygon> <!-- east wall -->
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList
      srsName="http://www.opengis.net/gml/srs/epsg.xml/#4979">
                       37.41188 -121.93132 0
                       37.41188 -121.93132 25
                       37.41142 -121.93132 25
                       37.41142 -121.93132 0
                       37.41188 -121.93132 0
                  </gml:posLis>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
            <gml:Polygon> <!-- south wall -->
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList
      srsName="http://www.opengis.net/gml/srs/epsg.xml/#4979">
                       37.41142 -121.93132 0
                       37.41142 -121.93132 25
                       37.41142 -121.93242 25
                       37.41142 -121.93242 0
                       37.41142 -121.93132 0
                  </gml:posLis>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
            <gml:Polygon> <!-- west wall -->
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList
      srsName="http://www.opengis.net/gml/srs/epsg.xml/#4979">
                       37.41142 -121.93243 0
```

```
                          37.41142 -121.93243 25
                          37.41188 -121.93243 25
                          37.41188 -121.93243 0
                          37.41142 -121.93243 0
                        </gml:posLis>
                      </gml:LinearRing>
                    </gml:exterior>
                  </gml:Polygon>
                  <gml:Polygon> <!-- roof -->
                    <gml:exterior>
                      <gml:LinearRing>
                        <gml:posList
       srsName="http://www.opengis.net/gml/srs/epsg.xml/#4979">
                          37.41188 -121.93243 25
                          37.41188 -121.93132 25
                          37.41142 -121.93132 25
                          37.41142 -121.93242 25
                          37.41188 -121.93243 25
                        </gml:posLis>
                      </gml:LinearRing>
                    </gml:exterior>
                  </gml:Polygon>
                </gml:patches>
              </gml:Surface>
            </gml:exterior>
          </gml:Solid>
        </my:Building>
      </enterOrExit>
    </location-filter>
```

If the resource enters or exits either the parking garage or any of the conference rooms (both of which are externally defined), send a notification:

```
    <location-filter>
      <enterOrExit>
        <my:ParkingGarage
    xlink:href="http://server.example.com/loc-defs/bldg-mgr/parking"/>
      </enterOrExit>
      <enterOrExit>
        <my:ConferenceRooms
    xlink:href="http://server.example.com/loc-defs/userdef/confrooms"/>
      </enterOrExit>
    </location-filter>
```

3.1 XML Schema for filter format

The XML Schema for this format is defined below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="urn:ietf:params:xml:ns:location-filter"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:gml="http://www.opengis.net/gml">

  <xs:element name="location-filter">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="movedHoriz" type="gml:MeasureType"
                    minOccurs="0" maxOccurs="1"/>
        <xs:element name="movedVert" type="gml:MeasureType"
                    minOccurs="0" maxOccurs="1"/>
        <xs:element name="speedExceeds" type="gml:MeasureType"
                    minOccurs="0" maxOccurs="1"/>

        <!-- this type needs to hold an XPath statement -->
        <xs:element name="valueChanges" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>

        <xs:element name="enterOrExit" type="gml:FeaturePropertyType"
                    minOccurs="0" maxOccurs="unbounded"/>

        <!-- Do we want to incldue this to allow new filters? -->
        <xs:any namespace="##other" processContents="lax"
           minOccurs="0"  maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4. Containment schema

This section describes the schema for describing the resource's location relative to a region or list of regions which might contain the resource. (These regions can be defined dynamically in an "enterOrExit" element in a subscription filter, or defined on the notifier using some out-of-band mechanism.) The "pidfResource" element is placed inside the location-info element in a PIDF-LO document. The pidfResource element can contain zero or more

"containment" elements. Each containment element has a GML Feature sub-element (of type "FeaturePropertyType") and a mandatory attribute which specifies if the PIDF resource is inside or outside of the feature, or if the position of the resource with respect to the region or region list is undefined. If the subscriber is not authorized to know the relative position, the notifier MUST NOT reveal this private information. The RECOMMENDED way to prevent the subscriber from seeing private location data of this type is to return a containment element whose position attribute is "undefined".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="urn:ietf:params:xml:ns:pidf:geopriv10:containment"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:gml="http://www.opengis.net/gml"
xmlns:pr="urn:ietf:params:xml:ns:pidf:geopriv10:containment" >
  <xs:element name="pidfResource">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="pr:containment"
             minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="containment">
    <xs:complexType>
        <xs:sequence>
          <xs:any namespace="http://www.opengis.net/gml"
                minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
      <xs:attribute name="position" use="required">
         <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="inside"></xs:enumeration>
            <xs:enumeration value="outside"></xs:enumeration>
            <xs:enumeration value="undefined"></xs:enumeration>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Below is an example PIDF-LO document which indicates that the resource is inside building 10, not outside the parking garage, and not permitted to know if the resource is in a conference room. Note that in GML, these features could be referenced by their unique

identifiers instead.

```
<?xml version="1.0" encoding="UTF-8"?>
  <presence xmlns="urn:ietf:params:xml:ns:pidf"
     xmlns:gp="urn:ietf:params:xml:ns:pidf:geopriv10"
     xmlns:pr="urn:ietf:params:xml:ns:pidf:geopriv10:containment"
     entity="pres:geotarget@example.com">
   <tuple id="sg89ae">
    <status>
     <gp:geopriv>
       <gp:location-info>
         <pr:pidfResource>
           <pr:containment position="inside">
            <my:Building>
               <gml:name>Building 10</gml:name>
             </my:Building>
           </pr:containment>
           <pr:containment position="outside">
             <my:ParkingGarage
xlink:href="http://server.example.com/loc-defs/bldg-mgr/parking"/>
           </pr:containment>
           <pr:containment position="undefined">
             <my:ConferenceRooms
xlink:href="http://server.example.com/loc-defs/userdef/confrooms"/>
           </pr:containment>
         </pr:pidfResource>
       </gp:location-info>
       <gp:usage-rules>
         <gp:retransmission-allowed>yes</gp:retransmission-allowed>
         <gp:retention-expiry>2003-06-23T04:57:29Z
         </gp:retention-expiry>
        </gp:usage-rules>
     </gp:geopriv>
    </status>
    <timestamp>2003-06-22T20:57:29Z</timestamp>
   </tuple>
  </presence>
```

5. Event Package Formal Definition

5.1 Event Package Name

This document defines a SIP Event Package as defined in RFC 3265 [2]. The event-package token name for this package is:

"location"

5.2 Event Package Parameters

This package does not define any event package parameters.

5.3 SUBSCRIBE Bodies

This package defines a SUBSCRIBE body format in Section 3, which is used to filter notifications. Subscribers MUST include a location filter with at least one filter event in every new or updated subscription request. (A filter is not necessary, nor desirable in an unsubscription request.)

5.4 Subscription Duration

Subscriptions to this event package MAY range from minutes to weeks. Subscriptions in hours or days are more typical and are RECOMMENDED. The default subscription duration for this event package is one hour.

5.5 NOTIFY Bodies

Both subscribers and notifiers MUST implement PIDF-LO. Notifiers MAY send location information in any format acceptable to the subscriber (based on the subscriber inclusion of these formats in an Accept header). "application/cpim-pidf+xml"

A future extension MAY define other NOTIFY bodies. If no "Accept" header is present in the SUBSCRIBE, the body type defined in this document MUST be assumed.

5.6 Subscriber generation of SUBSCRIBE requests

Each new subscribe request establishes a notification filter. Subsequent subscriptions keep the same filter unless a new filter is provided. If a new filter is provided in a subscription, it completely replaces the previous filter.

Subscriber User Agents will typically SUBSCRIBE to location information for a period of hours or days, and automatically attempt to re-SUBSCRIBE well before the subscription is completely expired. If re-subscription fails, the Subscriber SHOULD periodically retry again until a subscription is successful, taking care to backoff to avoid network congestion. If a subscription has expired, new re-subscriptions MUST use a new Call-ID.

The Subscriber MAY also explicitly fetch the current status at any time. The subscriber SHOULD renew its subscription immediately after a reboot, or when the subscriber’s network connectivity has just been re-established.

The Subscriber MUST be prepared to receive and process a NOTIFY with new state immediately after sending a new SUBSCRIBE, a SUBSCRIBE renewal, an unsubscribe, or a fetch; or at any time during the subscription.

5.7 Notifier processing of SUBSCRIBE requests

When a Notifier receives SUBSCRIBE messages with the location event-type, it SHOULD authenticate the subscription request. The Notifier MAY choose to provide very coarse location information to anonymous subscribers (ex: country, postal code, time zone). If authentication is successful, the Notifier SHOULD authorize the subscriber. In addition, the Notifier MAY provide different location granularity or obfuscation depending on the identity of the subscriber. If no location-filter is provided, the Notifier SHOULD reject the subscription with a 403 Forbidden response. The Notifier MAY further limit the duration of the subscription to an administrator defined amount of time as described in SIP Events.

For new subscriptions, or anytime the location-filter is updated by the subscriber, the notifier MUST include appropriate containment locations for every feature mentioned in an enterOrExit element in the corresponding filter. If the subscriber is not authorized to receive this information, the notifier MUST either include each these locations with the value of undefined, or alternatively, send a 403 Forbidden response to the subscriber.

5.8 Notifier generation of NOTIFY requests

Immediately after a subscription is accepted, the Notifier MUST send a NOTIFY with the current location information as appropriate based on the identity of the subscriber. This allows the Subscriber to resynchronize its state. When the location changes sufficiently to trigger any of the filter events in the current location-filter for the subscription, the notifier sends a notification with the new location information.

5.9 Subscriber processing of NOTIFY requests

The Subscriber MUST be prepared to receive NOTIFYs from different Contacts corresponding to the same SUBSCRIBE. (The SUBSCRIBE may have been forked).

5.10 Handling of Forked Requests

Forked requests are allowed for this event type and may install multiple subscriptions. Note that different Notifiers MAY provide (different) location information for different tuples. In this case,

multiple instances representing the same presentity have different locations.

In other cases, different Notifiers might provide different location for the same tuple. This presents an administrative problem. Certainly it is acceptable for me to express my location as "In San Jose, California, USA" and at specific coordinates or a specific address. Conventions for expressing multiple locations or multiple location formats are discussed in [9].

If all of the tuples contain information which is not contradictory, then this is not an error. If multiple notifiers provide contradictory information for the same tuple, this is an error. If multiple notifiers provide different tuples, or non-contradictory location information for the same tuple, this is not an error.

5.11 Rate of notifications

A Notifier MAY choose to hold NOTIFY requests in "quarantine" for a short administrator-defined period (milliseconds or seconds) when the location is changing rapidly. Requests in the quarantine which become invalid are replaced by newer notifications, thus reducing the total volume of notifications. This behavior is encouraged for implementations with heavy interactive use.

Notifiers SHOULD NOT generate NOTIFY requests more frequently than ten per second, nor more frequently than thirty in a thirty-second period of time.

5.12 State Agents and Lists

This document does not preclude implementations from building state agents which support this event package. Likewise, this document does not preclude subscriptions to lists of resources using the event list extension [14].

5.13 Behavior of a Proxy Server

There are no additional requirements on a SIP Proxy, other than to transparently forward the SUBSCRIBE and NOTIFY methods as required in SIP.

6. Examples of Usage

The examples shown below are for informational purposes only. For a normative description of the event package, please see sections 3 and 5 of this document.

In the example call flow below, the Subscriber subscribes to the status of the Notifier's location. Via headers are omitted for clarity. [TODO:]

7. Security Considerations

Location information is typically very privacy sensitive. At minimum, subscriptions to this event package SHOULD be authenticated and properly authorized. Furthermore, GEOPRIV requires that notifications MUST be encrypted and integrity protected using either end-to-end mechanisms, or the hop-by-hop protection afforded messages sent to SIPS URIs.

Implementations of this event package MUST implement the sips: scheme, and MUST implement the security requirements described in PIDF-LO [3]. In addition, all SIP implementations are already requried to implement Digest authentication.

Additional privacy and security considerations are discussed in detail in [9] and in SIP [1] and SIP Events [2].

8. IANA Considerations

8.1 SIP Event Package Registration for 'location'

Package name: location

Type: package

Contact: [Mahy]

Published Specification: This document.

8.2 MIME Registration for application/location-delta-filter+xml

MIME media type name: application

MIME subtype name: application/location-delta-filter+xml

Required parameters: none.

Optional parameters: none.

Encoding considerations: Same as for XML.

Security considerations: See the "Security Considerations" section in this document.

Interoperability considerations: none

Published specification: This document.

Applications which use this media: The application/location-delta-filter+xml application subtype supports the exchange of filters to throttle asynchronous notifications of location information in SIP networks.

Additional information:

1. Magic number(s): N/A

2. File extension(s): N/A

3. Macintosh file type code: N/A

8.3 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:location-filter

This section registers a new XML namespace, as per the guidelines in [13].
URI: The URI for this namespace is
urn:ietf:params:xml:ns:location-filter.
Registrant Contact: IETF, GEOPRIV working group, <geopriv@ietf.org>, as delegated by the IESG <iesg@ietf.org>.
XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Location Filter Namespace</title>
</head>
<body>
  <h1>Namespace for PIDF-LO Location Filters</h1>
  <h2>urn:ietf:params:xml:ns:location-filter</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
END
```

8.4 Schema Registration For location-filter

This specification registers a schema, as per the guidelines in in [13].

URI: please assign.

Registrant Contact: IETF, GEOPRIV Working Group (geopriv@ietf.org), as delegated by the IESG (iesg@ietf.org).

XML: The XML can be found as the sole content of Section 3.1.

8.5 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:pidf:geopriv10:containment

This section registers a new XML namespace, as per the guidelines in [13].

URI: The URI for this namespace is urn:ietf:params:xml:ns:pidf:geopriv10:containment.

Registrant Contact: IETF, GEOPRIV working group, <geopriv@ietf.org>, as delegated by the IESG <iesg@ietf.org>.

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>PIDF-LO Location Containment Namespace</title>
</head>
<body>
  <h1>Namespace for PIDF-LO location containment elements</h1>
  <h2>urn:ietf:params:xml:ns:pidf:geopriv10:containment</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
END
```

8.6 Schema Registration For containment

This specification registers a schema, as per the guidelines in in [13].

URI: please assign.
Registrant Contact: IETF, GEOPRIV Working Group (geopriv@ietf.org), as delegated by the IESG (iesg@ietf.orgw).
XML: The XML can be found as the sole content of Section 4.

9. Acknowledgments

Thanks to Allan Thompson, James Winterbottom, and Martin Thomson for their comments.

10. References

10.1 Normative References

[1] 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.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Peterson, J., "A Presence-based GEOPRIV Location Object Format", draft-ietf-geopriv-pidf-lo-03 (work in progress), September 2004.

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

[5] Bray, T., Paoli, J., Sperberg-McQueen, C., and E. Maler, "Extensible Markup Language (XML) 1.0 (2nd ed)", W3C REC-xml, October 2000, <http://www.w3.org/TR/REC-xml>.

[6] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, May 2001, <http://www.w3.org/TR/xmlschema-1/>.

[7] Bray, T., Hollander, D., and A. Layman, "Namespaces in XML", W3C REC-xml-names, January 1999, <http://www.w3.org/TR/REC-xml-names>.

[8] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", W3C Recommendation xpath, November 1999, <http://www.w3.org/TR/xpath>.

[9] Winterbottom, J., "GEOPRIV PIDF-LO Usage Clarification, Considerations and Recommendations", draft-winterbottom-geopriv-pdif-lo-profile-00 (work in progress), February 2005.

[10] OpenGIS, "Open Geography Markup Language (GML) Implementation Specification", OpenGIS OGC 02-023r4, January 2003, <http://www.opengis.org/techno/implementation.htm>.

10.2 Informational References

[11] Moats, R., "URN Syntax", RFC 2141, May 1997.

[12] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[13] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[14] Roach, A., Rosenberg, J., and B. Campbell, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", draft-ietf-simple-event-list-07 (work in progress), January 2005.

[15] Polk, J. and B. Rosen, "Session Initiation Protocol Location Conveyance", draft-ietf-sip-location-conveyance-00 (work in progress), June 2005.

Author's Address

Rohan Mahy
SIP Edge LLC

Email: rohan@ekabal.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 12, 2006

J. Rosenberg
Cisco Systems
C. Jennings
Cisco
J. Peterson
Neustar

July 11, 2005

Identity Privacy in the Session Initiation Protocol (SIP)
draft-rosenberg-sip-identity-privacy-00

Status of this Memo

Copyright Notice

Abstract

RFC3323 defines procedures for privacy in the Session Initiation Protocol (SIP). These mechanisms make use of a privacy service that resides in the network, which can remove identifying information from messages. Its approach to privacy was compatible with the identity mechanisms in RFC 3325, which defined the P-Asserted-ID header field.

However, its approach does not work well with the new cryptographic-based mechanisms in draft-ietf-sip-identity. As such, this document proposes a new framework for user privacy in SIP.

Table of Contents

1. Introduction

RFC 3323 [2] defines procedures for privacy in the Session Initiation Protocol (SIP). It provides guidelines for a UA to follow in the construction of its messages, so that identifying information is not placed into the message in the first place. However, it also defines a network-based privacy service that can be invoked by the client through the insertion of the Privacy header field. This privacy service typically runs within the user's default outbound proxy, and is responsible for removal of additional information from the messages. Two levels of privacy can be provided by this service - "header" privacy, which obfuscates identifying information from the SIP messages, and "session" level privacy, which includes the IP addresses used for exchange of media.

RFC 3325 [9], which defined the P-Asserted-ID header field, has seen widespread usage as the means for network authenticated identity in SIP. It defines another privacy service, the "id" service. This service causes elements in the network to strip the P-Asserted-ID header field when a request traverses a trust boundary.

RFC3325's form of identity has numerous drawbacks. Of these, the most significant is that the trustworthiness of the asserted identity is equal to the trustworthiness of the least trustworthy provider within the network of providers that constitute the trust domain. This works well in single provider environments, but in larger scale interconnects it eventually breaks apart. Unfortunately, the trustworthiness of an identity is a key property needed for nearly all of the VoIP anti-spam techniques [13]. For this reason, amongst others, [3] was developed. It provides strong cryptographic assurances of identity. It does so by providing a signature over the From header field in the request, and including in that signature information that provides referential integrity of the signature. This allows for recipients of the request to validate that the asserting domain has truly asserted the requestor's identity for that request. Since the mechanism is fundamentally domain-based, it also allows validating entites to apply policies regarding the trustworthiness of the asserting provider. This fundamentally avoids the "weakest link" property of RFC 3325.

There are numerous issues in the direct applicability of RFC 3323 to draft-ietf-sip-identity, many of which are pointed out in Section 13 of [3] (herein referred to as the "SIP identity specification" or the "SIP identity mechanism"). These problems are:

Intra-Domain Privacy: Because the SIP identity mechanism relies on the domain of the From header field as a key for obtaining certificates used to validate the identity in the From header field, anonymity is restricted to being within a domain. It is no longer possible, as described in RFC 3323, to populate the From header field with "anonymous.invalid" as the domain. As a consequence, a recipient of the request will be able to determine the domain of the originator of the request, though they will not be able to determine which user within that domain sent the request. This limitation is not very troubling for domains with extremely large numbers of users. However, for small domains, such as enterprises or home networks, it can be equally revealing as the identity of the requestor themselves.

Contact Privacy lost: Because the SIP identity mechanism relies on a signature over the Contact header field for referential integrity, a privacy service that provides header privacy cannot actually modify the Contact value. This will reveal the IP address of the requestor to the recipient of the request, which can often provide substantial information about the requestor.

Session Privacy lost: Session privacy is accomplished through a back-to-back user agent (b2bua) that rewrites the SDP to relay session media through an intermediary. This no longer works at all with the SIP identity mechanism, as it relies on a signature over the body of the request (which contains the SDP) to provide referential integrity.

Subscriber Identity Lost within Originating Domain: One of the benefits of the P-Asserted-ID header field when used in conjunction with the "id" level of privacy is that elements within the domain of the originator of the request will still be able to determine the identity of the originator. This is necessary for providing features for the requestor, accounting for their usage, and so on. With the SIP identity mechanism, if privacy is needed, the From header field contains an anonymous URI. As a result, the request has no information that can identity the user within their own domain, unless the SIP identity mechanism is used in conjunction with RFC3325, which is redundant.

These problems are in addition to the problems inherent in RFC 3323 to begin with:

Sensitivity to Boundary Configuration: Although RFC3323 argues strongly in favor of placing the privacy service very near the originator of the request, this goal is at odds with RFC 3325, which requires the privacy service to be on the egress edge of the trust domain. As a result, privacy is actually provided only if

every egress proxy is properly configured to take positive action and remove the P-Asserted-ID header field. Because positive action from the network is required to provide privacy, this mechanism is sensitive to misconfiguration of network elements, particularly in large interconnected trust domains.

Complicated Call Trace: In many networks, there is a requirement to provide a call trace feature that allows for malicious callers to be traced back to their source so that legal action can be taken. The utility of such features in a global SIP network aside, RFC 3323 makes such a feature difficult to provide since the identity of the requestor is literally removed from the request. This complicates the tracking procedures needed to identify the originator later on.

Limited Flexibility The degrees of privacy that RFC 3323 could provide were coded into the tokens valid in the Privacy header field. More complicated combinations - anonymity for certain media streams but not others, for example - were not possible.

This specification provides an alternate formulation for user privacy that works well in conjunction with [3]. This mechanism resolves nearly all of the limitations described above by moving more intelligence to the client, and having it act in cooperation with network services that provide atomic anonymity functions - IP address privacy via Traversal Using Relay NAT (TURN) [5] and URI privacy via an anonymous URI minting process.

2. Overview of Operation

When a user wishes to make an anonymous request, the user agent determines the set of identifying information that is to be obfuscated. This identifying information includes IP addresses, such as those in the Session Description Protocol (SDP) [6] and Via header fields, and URIs, such as those in the From header field and Contact header field of the request. User agents can anonymize any subset of this information in the request.

To anonymize IP addresses, the client contacts a TURN server [5], and obtains an IP address and port on the server which route to it. Ideally, this is done with a TURN server that is specifically dedicated to anonymous services, and thus can provide a higher degree of anonymity by obtaining anonymized IP address from a separate provider (see Section 6) than a normal one. The client uses the TURN-derived addresses in those fields of the message where the UA wishes to anonymize an IP address.

To anonymize URIs, and in particular the URI in the From header

field, the client needs to obtain a URI from its domain that possesses both the AOR property and the anonymity property (see [4] for a discussion of URI properties). To do that, it generates a special REGISTER request that effectively asks the provider to create a new URI for the user, and at the same time, register it. The network will construct this URI such that other network elements within the domain can use it to identify the requestor, but those outside cannot. This is readily done by creating the URI by encrypting the actual identity of the requestor combined with a large random number. Any element that shares the decryption key can know the identity of the user, but others cannot. In addition, the URI will have the "user" URI parameter present, and set to the value of "anonymous". This signals to all elements that the requestor is asking for anonymity. This is needed to prevent downstream elements within the domain from inserting additional identifying information, and also for properly rendering the fact that the caller was anonmyous.

The UAC then places this URI in the From header field of the request. It populates the Contact header field value with a Globally Routable User Agent URI (GRUU) [4] that was obtained through the registration which yielded the minted From URI. Beyond that, the other procedures of RFC 3323 around display names, Call-ID and other fields are followed.

This request is then sent into the network. There is no Privacy header field or other network involvement needed in order to further anonymize the request. Within the domain of the originator, proxy servers that see that the From header field contains an anonymous URI can decrypt it to obtain the identity of the requestor. Of course, elements outside of the domain will not possess the key, and therefore will not know the identity of the requestor. Because positive action is required in the network to obtain their identity (namely, acquisition of the decryption key and decryption of the URI), the mechanism is privacy-safe. Network misconfiguration can, in the worst case, result in a proxy not determining the identity of the requestor.

Furthermore, since the From field URI is carried all the way to the recipient of the request, it is possible to "call them back", even though the request was anonymous. Of course, the originating domain can decide to reject such requests, but this becomes a matter of local policy. The fact that the identity of the requestor, suitably encrypted, is carried all the way to the recipient of the request also facilitates services like malicious call trace. A network provider can contact the domain administrator of the domain on the right hand side of the at-sign, and request decryption of the user part in order to identify the malicious caller. Since these requests

are handled off-line and not in real time, they can be suitably authorized.

3. UAC Behavior

3.1 Determining the Level of Anonymity

When a user wishes to send a request, whether it is an INVITE to initiate a session, or a SUBSCRIBE [10], MESSAGE [11] or any other method, the UA makes a determination about the level of anonymity that is desired. Typically, this would be based on user input or on local configuration or policy. The precise means for making this determination is outside of the scope of this specification.

Ultimately, however, the level of anonymity is expressed as a function of which types of identifying information (IP address, hostname, URI or display name) are to be anonymized, and in which fields of the SIP message. The following fields typically contain identifying information about the user:

From: This field contains the identity of the requestor, and will be signed by an identity service within the domain of the requestor. As such, clients desiring anonymity SHOULD populate this with a URI obtained through the procedures of Section 3.2. The display name also contains identifying information. It is RECOMMENDED that this be omitted when the requestor requires anonymity. This is a change from RFC 3323, which recommended a value of "Anonymous". Rather than relying on a display name to indicate an anonymous call, which is language-specific and not meant for consumption by an automata, the "user" URI parameter of the From header field indicates that the request was anonymous.

Contact: This field contains a URI used to reach the UA for mid-dialog requests and possibly out-of-band requests, such as REFER [12]. It is RECOMMENDED that this field be populated with the GRUU obtained through the minting procedures of Section 3.2. The display name also contains identifying information. It is RECOMMENDED that this be omitted when the requestor requires anonymity.

Reply-To: This field contains a URI that can be used to reach the user on subsequent call-backs. Clients desiring anonymity SHOULD populate this with a URI obtained through the procedures of Section 3.2. The display name also contains identifying information. It is RECOMMENDED that this be omitted when the requestor requires anonymity.

Via: This field contains an IP address and port that is used to reach the user agent for responses. It is RECOMMENDED that this field be populated with an IP address and port learned through a TURN server Section 3.3.

Call-Info: This field contains additional information about the requestor. It is RECOMMENDED that this field be omitted from requests.

Call-Info: This field contains additional information about the requestor's user agent. It is RECOMMENDED that this field be omitted from requests.

Organization: This field contains additional information about the requestor. It is RECOMMENDED that this field be omitted from requests.

Subject: This field contains freeform text about the subject of the call. Since it is not possible to know what content a user has inadvertently placed into such a header field, it is RECOMMENDED that this field be omitted from requests.

Call-ID: User agents SHOULD substitute for the IP address or hostname that is frequently appended to the Call-ID value a suitably long random value (the value used as the 'tag' for the From header of the request might even be reused).

SDP c/m lines: The c and m lines in the SDP body convey an IP address and port for receiving media. It is RECOMMENDED that this field be populated with an IP address and port learned through a TURN server Section 3.3.

SDP o line: The username SHOULD be set to "-". The IP address in this field SHOULD be populated with an IP address and port learned through a TURN server Section 3.3.

SDP s line: The session name SHOULD be set to "-".

SDP i,u,e,p lines: These lines SHOULD be omitted from the SDP.

3.2 Minting an Anonymous AOR

A key aspect of this specification is the ability of a UA to obtain an anonymous URI for placement into the From and Reply-To header fields, along with a GRUU that can be placed into the Contact header field. It is RECOMMENDED that the UA obtain a new anonymous URI for each new request outside of an existing dialog that it generates.

To obtain a new URI that is suitable for placement into the From header field of a new request, a UA constructs a query REGISTER request according to the procedures of RFC 3261. This request is not anonymous; a UA MUST correctly populate the To, From and other header fields of the request. This request MUST utilize the GRUU mechanism, and thus include the Supported header field with the value "gruu" [4]. The Contact header fields, however, are omitted as this is a query registration. However, the UA MUST include the Require header field with the option tag "anonymous". This instructs the registrar to view this request as a special query; one that provides the UA with a brand new set of anonyous URIs that represent aliases for the user's AOR and registered contacts.

The REGISTER response will contain the set of currently registered Contacts against the AOR in the To header field. In addition, the response will contain the Anonymous-To header field. This header field will contain a URI that has both the AOR and anonymous properties, and which represents an alias of sorts for the user's actual AOR. Its not a pure alias, in that requests sent to that URI don't get equivalent treatment to requests sent to the AOR. Domain policy may result in different treatment for requests made to that URI. This specification provides no automated means for the user to request specific policies. The URI from the Anonymous-To header field can be placed into the From and Reply-To header fields of an outgoing request. Note that each and every REGISTER transaction sent by the client with the "anonymous" option tag in the Require header field will mint a new anonymous URI in the Anonymous-To header field.

In addition, because the client had indicated support for the GRUU mechanism, the REGISTER response will also contain a GRUU for each registered contact. However, these GRUU will also be freshly minted, and have the anonymous property as well as the GRUU property. Like Anonymous-To, each REGISTER transaction produces a new set of GRUU in the Contact header field of the REGISTER response. The client then uses the GRUU for its own instance in the Contact header field of a request.

3.3 Obtaining an Anonymous IP Address

To obtain an anonymous IP address and port for usage in the SDP, Via header field and other parts of the SIP message, a client contacts a configured TURN server [5]. It uses normal TURN processing to allocate those addresses. Local policy in the TURN server will produce IP addresses and ports with poor correlation properties, as discussed below.

4. Registrar Behavior

A registrar compliant to this specification MUST support the GRUU specification in addition to this one.

When the registrar receives a REGISTER request, it checks for the presence of the Require header field. If present, and if it includes the option tag "anonymous", processing follows as described in this section.

If the REGISTER request contains any Contact header fields, the registrar MUST reject the request with a 403. REGISTER requests that mint anonymous URIs have to be query registrations. As such, the registrar follows normal RFC3261 and GRUU processing for constructing the response.

Next, the registrar generates an anonymous URI that has the AOR and anonymous properties. This URI can be within the domain of the provider, however, ideally it is within a domain or set of domains set aside explicitly for anonymous URI. See Section 6. This specificaiton makes no normative recommendations on how such a URI is constructed. However, it MUST have the following properties:

- The user part has at least 256 bits of randomness.
- There is no correlation possible between two URIs given to the same user.
- Network elements within the domain of the user, to whom explicit keying material has been granted, can extract the actual AOR of the user from the URI.
- The URI MUST include the URI "user" parameter with the value "anonymous".

One simple way to obtain a URI with these properties is to form the user part of the URI by encrypting the AOR of the subsciber concatenated with 256 bits of random salt.

Once done, the registrar places this URI in the Anonymous-To header field of the REGISTER response. Furthermore, it takes each GRUU present in the Contact header fields of the REGISTER response, and replaces them with an anonymous URI that has the following properties:

- The user part has at least 256 bits of randomness.

- There is no correlation possible between two URIs given to the same user.

- Network elements within the domain of the user, to whom explicit keying material has been granted, can extract the actual GRUU of the user from the URI.

- The URI MUST include the URI "user" parameter with the value "anonymous".

A domain MAY confer other properties upon the Anonymous-To and GRUU URI. In particular, it is expected that the service treatment property would be applied, though the services invoked for incoming requests to that URI would likely be different. It is expected that services like special call logs, or time-based call blocking, would be applied.

5. Proxy Behavior

A proxy that receives a request whose From header field has a URI whose user parameter has the value "anonymous", but needs to know the identity of the requestor for processing, SHOULD attempt to extract the AOR from the URI in the From header field based on domain-specific procedures. [[OPEN ISSUE: for multi-vendor SIP networks within a single domain, do we require these algorithms to be standardized?]]

When a proxy compliant to this specification sees a request whose From header field has a URI whose user parameter has the value "anonymous", it MUST NOT insert additional information into the request that identifies the originator of the request, if the originator is known to the proxy. Besides the header fields listed in Section 3.1, the Path [7], Service-Route [8] and Record-Route header fields are inserted by proxies and often contain identifying information.

6. Anonymity Providers

Note - this section is likely to be highly contentious and it is also highly speculative. It is readily extracted from the rest of the specification and it provides the mechanisms necessary for the highest levels of anonymity.

Since the mechanism defined in this specification is meant to be compatible with [3], it relies on domain-based signatures. As such, identity is always within the scope of a domain that will be known to the recipient of the request. Similarly, IP addresses obtained from TURN servers will be within the IP address space of the provider of

the server. Unfortunately, the allocations of IP addresses to providers is a well-known property, and thus the provider can often be determined from examination of the IP address. As discussed above, simply knowing the provider of the user sending the request can reveal substantial information about the requestor.

To deal with this, this specification recommends the creation of special providers called "anonymity providers". These are large providers (indeed, ideally there is a single one for the Internet), whose sole responsibility is to obtain and delegate names and addresses to actual providers using randomized allocation procedures. Actual SIP providers would contract with the anonymity provider under some form of agreement.

An anonymity provider would obtain a relatively large block of IP addresses from IP address blocks throughout the Internet. When a SIP provider is asked by one of its own customers to allocate an IP address and port for the purposes of anonymous calling, the TURN server that has received the request will obtain an IP address from the anonymity provider. This can be done in many ways. The simplest way is to have the SIP providers TURN server send a TURN request to the anonymity provider's TURN server, which then chooses one of its large number of addresses randomly. This approach has the drawback of funneling traffic through the anonymity provider. A more interesting approach is to have the SIP providers, on a daily or hourly basis, literally lease a block of addresses from the anonymity provider, and then inject BGP routes into the Internet for that address block. In this case, the anonymity provider serves the role of coordinator, making sure it is clear which SIP provider owns that particular block of IP addresses at any point in time. That avoids injection of the prefix into BGP from duplicate providers.

Similarly, the anonymity provider would ideally own a TLD (.anonymous, for example), act as a root CA, and be capable of creating sub-domains within this TLD. On a daily or hourly basis, each SIP provider would be given a new sub-domain whose value was newly minted and randomized (for example, h77asff-dg98asdkjkasdpapiasdddd.anonymous), along with certificates that would allow a SIP provider to sign requests with that domain. All SIP endpoints would possess the root CA certificate for the anonymity provider (which is why there can't be too many of them).

For this approach to work, automated protocols need to be put in place for the assignment of IP address blocks, subdomains in the anonymous TLD, and domain certificates within those subdomains. Future work is needed to define the protocols appropriate for such procedures.

Presumably, such an anonymity provider would be required to maintain the strictest standards of process and security, in order to provide high levels of anonymity in concert with the necessary levels of audit and tracing when government authorities require it. For this reason, it would seem likely that these anonymity providers would be country specific, though it need not be the case.

It should be further noted that such an anonymity provider is providing services that aren't specific to SIP, and could be utilized by any application provider that wishes to provide anonymous services to its own customers. It would allow, for example, anonymous email or anonymous instant messaging services, or anonymous web browsing.

7. Grammar

This specification defines a new header field, Anonymous-To, a SIP option tag, anonymous, and a new value of the user parameter of the SIP URI:

```
Anonymous-To    =     "Anonymous-To" HCOLON ( name-addr / addr-spec )
                *( SEMI generic-param )
anonymous-tag   =     "anonymous"
user-param      =  "user=" ( "phone" / "ip" / "anonymous" /
                      other-user)
```

8. Examples

TODO.

9. Security Considerations

This specification is intimately concerned with issues of security. A nice summary needs to go here.

10. IANA Considerations

This specification registers a new SIP option tag, a new SIP header field, and a new value of an existing URI parameter. Those registrations will go here.

11. References

11.1 Normative References

[1] 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.

[2] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[3] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[4] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-03 (work in progress), February 2005.

[5] Rosenberg, J., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-07 (work in progress), February 2005.

[6] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[7] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[8] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration", RFC 3608, October 2003.

11.2 Informative References

[9] Jennings, C., Peterson, J., and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[10] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

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

[12] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[13] Rosenberg, J., "The Session Initiation Protocol (SIP) and Spam", draft-ietf-sipping-spam-00 (work in progress), February 2005.

Authors' Addresses

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Cullen Jennings
Cisco
170 West Tasman Dr.
San Jose, CA 95134
US

Phone: +1 408 527-9132
Email: fluffy@cisco.com

Jon Peterson
Neustar
1800 Sutter Street
Suite 570
Concord, CA 94520
US

Phone: +1 925 363-8720
Email: jon.peterson@neustar.biz
URI: http://www.neustar.biz

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 12, 2006

J. Rosenberg
Cisco Systems

July 11, 2005

Clarifying Construction of the Route Header Field in the Session Initiation Protocol (SIP)
draft-rosenberg-sip-route-construct-00

Status of this Memo

Copyright Notice

Abstract

The Route header field in the Session Initiation Protocol (SIP) protocol is used to cause a request to visit a set of hops on its way towards the final destination. The SIP specification defines construction of the Route header field at user agents. However, numerous other mechanisms have been described, such as Service-Route and the 305 response, which cause the client to set its Route header field for a request. As such, the specific behavior for a UA in construction of its Route header field is unclear. This document

attempts to define a consistent set of logic.

Table of Contents

1. Introduction

The Route header field in the Session Initiation Protocol (SIP) protocol is used to cause a request to visit a set of hops on its way towards the final destination. RFC 3261 [2] discusses how a client constructs the Route header field for requests. However, this logic is restricted to mid-dialog requests, where the route set was learned as a result of record-routing.

However, additional sources of routes can exist for a UA. These include default outbound proxies, a service route learned from the Service-Route header field [3], and a redirection coming from a 305 response. In total, there are four sources of potential route headers. The way in which these various sources are reconciled is unclear. Furthermore, the various specifications are unclear about which requests these Route headers are applicable to. Do they apply to REGISTER? Do they apply to mid-dialog requests?

Section 2 reviews the existing sources of route sources. Section 3 discusses problems with the existing specifications. Section 4 overviews the proposed changes in behavior. Section 5 provides a detailed description of element behavior, and Section 6 discusses backwards compatibility issues.

2. Existing Sources

This section examines the current set of route header field sources.

2.1 Default Outbound Proxies

RFC 3261 discusses default outbound proxies. In Section 8.1.1.1, it makes reference to its interaction with Route header fields:

> In some special circumstances, the presence of a pre-existing route set can affect the Request-URI of the message. A pre-existing route set is an ordered set of URIs that identify a chain of servers, to which a UAC will send outgoing requests that are outside of a dialog. Commonly, they are configured on the UA by a user or service provider manually, or through some other non-SIP mechanism. When a provider wishes to configure a UA with an outbound proxy, it is RECOMMENDED that this be done by providing it with a pre-existing route set with a single URI, that of the outbound proxy.
>
> When a pre-existing route set is present, the procedures for populating the Request-URI and Route header field detailed in Section 12.2.1.1 MUST be followed (even though there is no dialog), using the desired Request-URI as the remote target URI.

The default outbound proxy can be learned either through DHCP [4], through configuration (such as the SIP configuration framework [6]), or through other means. In the IMS, the default outbound proxy is the P-CSCF and is learned through GPRS specific techniques.

RFC 3261 does not explicitly say the set of messages to which this route set applies. However, the text above implies that it is for all requests outside of a dialog.

2.2 Service Route

RFC 3608 specifies the Service-Route header field. This header field is provided to the UA in a 2xx response to a REGISTER request. The client uses this to populate its Route header fields for outgoing requests. However, RFC 3608 explicitly says that the decision a UA makes about how it combines the service route with other outbound routes is a matter of local policy. Furthermore, RFC 3608 does not clearly define to which requests the service route applies, and in particular, whether or not it applies to a REGISTER request or a mid-dialog request.

2.3 Record-Routes

RFC 3261 provides a detailed description of the record-routing mechanism, and how the user agents in a dialog construct route sets from the Record-Route header field values. RFC 3261 is also clear that the resulting route set applies to mid-dialog requests. It implies (though does not explicitly say) that the resulting route set overrides any default outbound proxies (which represent a pre-loaded route set).

2.4 305 Use Proxy

RFC 3261 defines the 305 "Use Proxy" response code, but says extremely little about exactly how it is used. It has this to say:

> The requested resource MUST be accessed through the proxy given by the Contact field. The Contact field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 (Use Proxy) responses MUST only be generated by UASs.

It is unclear how the Contact in the redirect is used. Does it populate the request URI of the resulting request? Or, does it get used to populate the Route header field? The restriction to UASs is also not explained.

Historically, the reason for the restriction to UAs was to avoid routing loops. Consider an outbound proxy that generates a 305,

instead of proxying the request. The concern was that the client would then recurse on the response, populate the Contact into a new request URI, and send the request to its default outbound proxy, which redirects once more. To avoid this, the RFC says that only a UAS can redirect with a 305, not a proxy.

However, this design decision on 305 handling was made prior to the conception of loose routing, although both ended up in RFC 3261. The design of the 305 mechanism, unfortunately, was not revisited after loose routing was specified. As such, the draft is not clear about whether or not the contact gets utilized as a Route header field value or whether it replaces the Request URI.

3. Problems with Current Specifications

Because the interactions between these various sources of routes are unspecified, certain features have proven impossible to provide, and/or interoperability problems have resulted.

One problem is that, depending on the way a client constructs its route set, it may be impossible to change a users outbound proxy without updating its configuration. Such changes are extremely useful for many operational reasons. One example is movement of subscribers between geographically distributed sites in cases where a site must be gracefully taken out of service, and the subscribers using it need to be moved. If the client uses the service route to augment the route from corresponding to its default outbound proxy, a network provider cannot move a subscriber.

Another problem is the client bootstrapping problem. Consider the same SIP network that utilizes geographically distributed sites. Each site contains a subset of the user database - the subset for the users in that site. When a SIP UA first boots up, it needs to obtain its configuration. As such, it has a hard-coded default proxy it uses for an initial SUBSCRIBE to enroll in its configuration [6]. This proxy, however, may not be the one in site to which the user of that SIP UA is associated. Ideally, the initial SUBSCRIBE could be routed to a server that redirects the client to the right proxy in the user's actual site. This redirection needs to override the default outbound proxy for the phone. However, there is not currently a way to do that.

An interoperability problem that has arisen is keeping an outbound proxy on the path for outbound requests. Consider a proxy in a hotel which a client discovers via DHCP and uses as its outbound proxy. This proxy wishes to be used for incoming and outgoing requests, both in and out of a dialog. So, it includes itself on the Path header field of the REGISTER. However, it has no idea if the registrar will

reflect the Path header field into the Service-Route, and cannot determine whether putting itself on the Path is effective for getting on the service route. Per RFC 3608 it cannot modify the Service-Route in the response to REGISTER. As such, if the registrar does not include the proxy in the Service-Route, and the endpoint overrides its outbound proxy setting with the Service-Route, the local proxy falls off the outbound path despite its best efforts.

4. Overview of Operation

Firstly, new behavior for generation and processing of the 305 Use Proxy is specified. Any element in the network, proxy or UAS, can generate a 305, not just a UAS as specified in RFC3261. This redirect can be recursed by any upstream element, but it is ideally recursed by the element directly upstream from the one that genreated the redirect. To recurse on the redirect, the proxy or UAC takes the Contact header field value from the 305, and uses it to replace the top value of the Route header field used previously. If no Route header field was used previously, one is added. However, in neither case is the Request-URI modified.

When a UAC goes to send a request, whether it is a mid-dialog request or a new request with any method (except CANCEL or ACK to a non-2xx response), the client first uses any route set learned from a record-route (which covers mid-dialog requests). If the request is not a mid-dialog reuqest, the client sees if it has any service routes learned through RFC 3608. If there are none, the client next uses any configured default outbound proxies. These three sources - record-routes, service routes and default outbound proxies - are never mixed, and one and only one of them applies to each request. After it is applied however, if the request results in a 305 Use Proxy response, the topmost Route header field is updated as described above.

A registrar, upon receipt of a REGISTER, uses the Path header field values to construct the Service-Route in the response. The values from the Path are copied into the Service-Route, and the registrar can then add some additional ones if they are within the domain of the provider.

5. Detailed Processing Rules

5.1 Registrar Behavior

The registrar MUST construct the Service-Route in the registration response by taking each URI from the Path header field in the REGISTER request, and inverting the order. After inversion, the registrar MAY add additional URIs at the end of the list (that is,

the right hand side of the list, corresponding to proxy elements that will be the farthest away from the UA).

Furthermore, the registrar MAY replace or remove any URIs that are within a domain under the control of the registrar. When replacing a URI, one or more new ones can take its place. If the registrar is in example.com, this would include any URIs whose domain part is example.com. It would also include any URIs whose domain is a subdomain of example.com, as long as that subdomain is under the control of example.com. It could also include URIs whose domain part is unrelated to example.com, as long as those are within the control of example.com. It is difficult to provide a concise definition of "under the control", but generally it means that the administrative policies for the subservient domain are completely defined by the controlling domain.

This behavior ensures that proxies outside of the domain of the registrar have a way to appear on the service route, but provides a way, within a domain, to provide service routes that are not coupled to the Path.

5.2 UAC Behavior

A UAC compliant to this specification MUST include the "lr305" option tag in the Supported header field of requests that it generates.

For a request sent by a UAC that is not the result of recursion on a 305, the following logic MUST be used to compute the route set used to populate the Route header field of the request. If the request is a mid-dialog request, the route set is computed per the procedures in Section 12.2.1.1 of RFC 3261. This route set overrides routes learned from configuration, DHCP, Service-Route or any other mechanism. If the request is not a mid-dialog request, the client checks to see if it has learned a service route as a result of registering the AOR it has populated in the From header field of the request. If it has learned a service route, the URIs from the Service-Route header field is used as the route set for the request. This route set overrides routes learned from configuration, DHCP, or any other mechanism. This route set is used in all requests outside of a dialog, including REGISTER. If the UA has not learned a service route, it uses the route set learned through configuration. [[OPEN ISSUE: Do we need to specify how to reconcile route sources learned across disparate configuration sources? For example DHCP and a config file?]]

5.3 Client Behavior

The following logic defined here applies to all clients, both UAC and

proxies, and applies to the processing of a 305 response.

It is RECOMMENDED that a client in receipt of a 305 recurse on that redirection, rather than forwarding it upstream. To compute the request that is sent as a result of the recursion, the client MUST take the route set used for the request that generated the 305 response. If that request had a Route header field, the first value MUST be replaced with the value of the Contact header field in the 305 with the highest q-value. If there are multiple such Contacts with the same q-value, one is chosen at random. The result is used as the route set for the new request. If the original request did not have a Route header field, the new request MUST contain a single Route header field value, equal to the URI provided in the Contact header field of the 305 with the highest q-value. This processing applies to requests both inside and outside of a dialog, and applies to all request methods, including REGISTER, with the exception of ACK and CANCEL.

If a 305 response had multiple Contact header field values, and the recursed request generated a 503 response, and the client had exhausted all alternative servers learned from DNS [5] for the previous Contact header field value, the client SHOULD choose the Contact from the 305 with the next highest q-value, and construct another recursed request using the procedures defined above. In the event the 305 had multiple Contact header field values with equivalent q-values, the next highest one might have a q-value equal to the one that was just tried.

If the policy of the client is such that it a request must visit a particular set of hops subsequent to being processed, and the route set constructed as a result of the recursion does not meet those policy constraints, the client MAY push additional route header field values in order for the request to meet those policy requirements. Proxies that do this SHOULD verify that the URI placed into the topmost Route header field value is an acceptable next hop, and not just blindly push route header field values.

5.4 Server Behavior

Any server, either a UAS or a proxy, MAY generate a 305 in response to a request. Such a response can be generated either for initial or mid-dialog requests. The 305 SHOULD NOT be generated unless one of the following conditions is met:

- The server generating the 305 has an administrative relationship with the previous hop element, and knows that it is capable of supporting this specification and will recurse on a 305 that it sends.

- The server generating the 305 believes that its previous hop is a UAC, and the request being redirected included a Supported header field with the option tag "lr305".

These requirements provide a limited form of backwards compatibility. See Section 6 for a thorough discussion.

6. Backwards Compatibility

This specification defines a different behavior for the processing of 305 than is implied in RFC 3261 (although the behavior is not entirely clear). Because of this, there are two backwards compatibility scenarios that need to be considered:

1. The element that recurses on the redirection does not support this specification. As a result, it replaces its Request-URI in the recursed request with the value from the Contact header field of the 305.

2. The element that recurses supports this specification, and correctly populates the Contact header field value into the Route header field of the recursed request. However, the element that performed the recursion was not the element immediately upstream from the one that generated the 305. As a result, an intermediate element is bypassed even though the desire was for it to remain on the route set.

The first of these two cases causes the Request URI to be clobbered. The request will arrive at the server that was the target of the redirection, but it probably won't be able to process the request because the actual request URI is no longer present. Unfortunately, avoiding this failure case entirely is quite difficult. It requires the redirecting server to have an assurance that the element immediately upstream, whether it is a proxy or UAC, supports this specification. There is no mechanism in the suite of RFC 3261 compatibility tools that can provide such a function. The only way to do this is to include another cookie in the Via branch ID, used as a signal that this extension is supported. However, this results in substantial pollution of the Via header field, and increases each message substantially.

It is believed that a 305 redirection is in fairly limited usage at the time of writing, and so this specification provides a weaker form of backwards compatibility. The Supported header field is used to verify that clients support the mechanism. Rather than explicit signaling, it is assumed that proxies can know whether the previous hop supports this mechanism based on an administrative relationship with that proxy. This precludes 305 from being used inter-provider

until it is ubiquitously deployed. However, this does not seem like a major limitation, since most of the use cases are intra-provider. The backwards compatibility mechanism also assumes that a proxy can determine that its previous hop is a UAC as opposed to a proxy; this is hard to know for certain.

The second backwards compatibility issue is interesting. What happens if the 305 is properly handled, but is recursed by an element that lies multiple hops upstream from the redirecting server? The recursing element will replace its top Route header field with the value from the Contact in the 305, and presumably send the request there directly. That may or may not be a problem, it depends on whether the previously-intervening proxies really need to be on the request path or not. To deal with this case, the specification allows a recursing element to push additional route headers in order to make sure requests traverse paths that meet their policy constraints.

7. Security Considerations

An attacker that injects a fake route set, whether it is in a 305 response, a Service-Route, a Record-Route or a configuration, can launch a multitude of attacks, including denial-of-service and fraud. For this reason, an element SHOULD NOT make use of a route set unless it has obtained it through a signaling channel that has been secured using the SIPS mechanism in RFC 3261 [2]

8. IANA Considerations

This specification registers a new option tag for SIP, according to Section 27.1 of RFC 3261.

Name: lr305

Description: This option tag is for support of the loose routing behavior for the 305 Use Proxy response. It is used in the Supported header field of requests, and indicates that the UAC will properly recurse when it receives a 305.

9. Acknowledgements

The author would like to thank Paul Kyzivat and Anders Kristensen for their comments.

10. References

10.1 Normative References

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

[2] 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.

[3] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration", RFC 3608, October 2003.

10.2 Informative References

[4] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, August 2002.

[5] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[6] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-06 (work in progress), February 2005.

Author's Address

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING
Internet-Draft
Expires: January 14, 2006

J. Rosenberg
Cisco Systems

July 13, 2005

Registration Coupled Subscriptions in the Session Initiation Protocol (SIP)
draft-rosenberg-sipping-reg-sub-00

Status of this Memo

Copyright Notice

Abstract

When a Session Initiation Protocol (SIP) user agent starts up, it registers to the network and initiates numerous subscriptions in order to learn about various network events. This results in a chatty startup procedure which substantially impacts recovery times under avalanche restart. This specification proposes a mechanism whereby the subscriptions can be established as a side effect of the registration, alleviating this problem.

Table of Contents

1. Introduction

When a Session Initiation Protocol (SIP) [1] user agent starts up, it typically follows a series of message exchanges with servers in the network. At a minimum, this startup procedure involves a SIP registration that allows the user agent to receive incoming requests. However, over time, numerous event packages [2] have been defined that provide a user agent with useful information through the duration of its connection to the network. These packages include:

Message Waiting: RFC 3842 [11] provides a message waiting indication event package. Typically, a user agent would subscribe to its own Address-of-Record (AOR) for this event package, in order to find out about messages that have been left for that user. This provides the familiar "message waiting lamp" on many business telephones. It is valuable for a user agent to subscribe to this package through the duration of its registration, in the event that messages are explicitly directed to a user's voicemail and do not ring their phone (this can happen, for example, if the caller utilizes the caller preferences specification [12] to direct a call to voicemail).

Registration Event: RFC 3680 [13] allows a user agent to learn about the status of its registration. Typically, a user agent would subscribe to its own AOR for this event package, in order to find out if the network has removed its registration. Such removals happen in cases of graceful network shutdown, or when a user needs to re-register and re-authenticate due to concerns on validity of credentials.

Presence List: A user may have a "buddy list", which contains a list of users whose presence is desired. A user will subscribe to their buddy list using an event list subscription [14] to the presence event package [15]. This is done by subscribing to a resource that is synonomous with the user's own buddy list.

Watcher Info: In order to find out about attempts that have been made to subscribe to a users presence, that user makes use of the watcher info event template package [16]. They would do this by subscribing to their own AOR with the presence.winfo event package. Subscription attempts that are unauthorized will result in a notification, informing the user of this fact and allowing them to approve or deny the subscription.

Dialog Events: Certain features, such as single line extension, require a user agent to find out about calls in progress on other user agents associated with the same AOR. This is done through subscriptions to the dialog event package [17]. The user agent

would typically subscribe to their own AOR, and learn about calls in progress to other user agents.

Configuration Events: The configuration event package [18] allows a UA to learn about changes in its configuration. This is done by having the UA subscribe to its own identity (which may be the AOR) for the config event package.

As a consequence of this, each time a user agent starts up, they will generate a REGISTER transaction, plus a SUBSCRIBE and a NOTIFY transaction for each event package the user agent is interested in. Based on the above discussion, this could be upwards of six event packages, resulting in a total of fourteen transactions that take place on startup. Furthermore, each of these subscriptions needs to be periodically refreshed (as does the registration), resulting in ongoing messaging.

This overhead is particularly problematic during an avalanche restart. This occurs when a failure event of some sort causes all user agents to simultaneously re-register. This is most common when recovering after a power outage. When the power returns, all the user agents will start booting simultaneously, and at the same time, each will execute their startup sequence. The more complex this sequence, the longer it takes for the system to return to service, and the more robust the network has to be. Another cause of avalanche restart is recovery after a catastrophic network failure, such as a network partition. If a network partition should last longer than the subscription lifetime, once the partition heals, each client will discover this and attempt to re-register and re-subscribe to each event package.

The overhead is also problematic on wireless links and other interfaces where bandwidth is at a premium.

2. Requirements

A solution to this problem should meet the following requirements:

1. The solution must substantially reduce the amount of SIP messaging traffic that takes place when a user agent starts up.

2. The solution must substantially reduce the amount of network processing that needs to take place when a user agent starts up.

3. The solution must not fundamentally alter the event model of RFC3265.

3. Proposed Solution

This document proposes a solution to this problem, based on the following observations:

1. In all of the above cases, the subscription is desired for the duration of the registration of the UA.

2. In all of the above cases, the user agent is subscribing to a resource which it owns; either its AOR or a related resource, like a buddy list. As a consequence, the authorization policies for the subscriptions always allow that user to subscribe. A policy in which a user can subscribe to their own events are called "self authorization".

3.1 Overview of Operation

Based on these observations, the approach proposed here is to strongly couple subscriptions with registrations, and to actually use the registration to create the subscriptions. A subscription that is created as a result of a successful registration is called a registration-coupled subscription. The basic approach is shown in

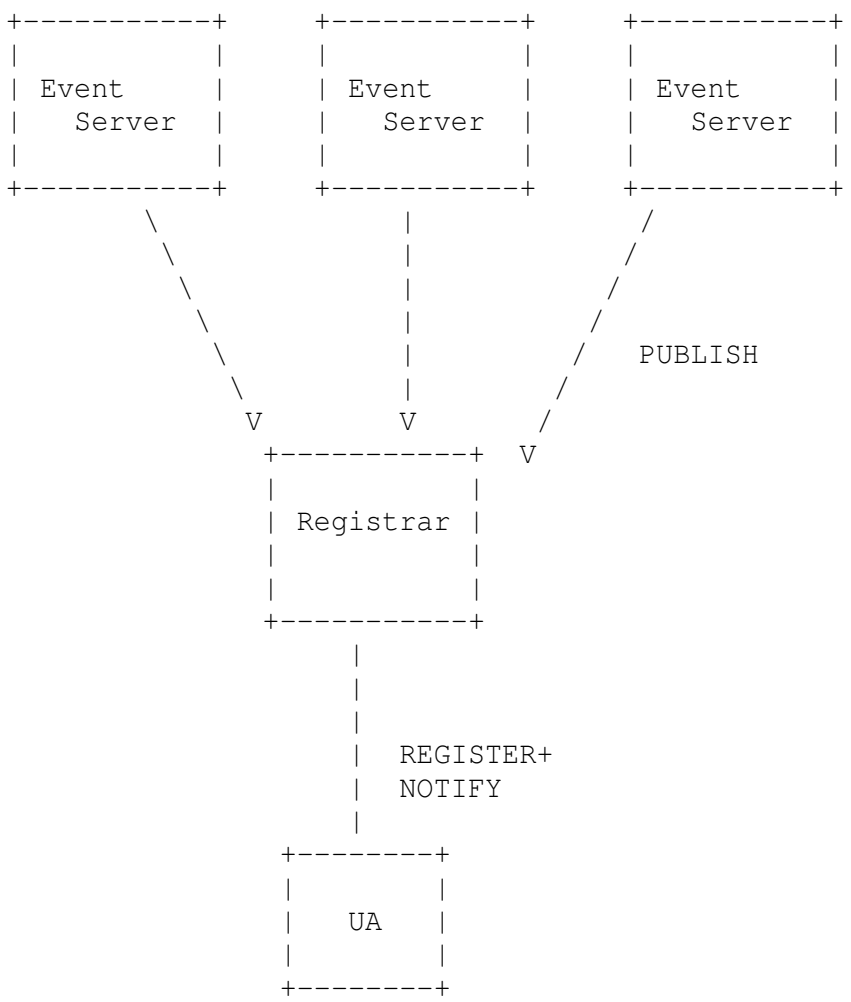


Figure 1

To create a registration-coupled subscription, a UA includes a Subscription header field in its REGISTER message. This header field includes a list of the desired event packages, and for each, the resource to which a subscription is desired and any event header field parameters. There is no need for a Require header field. The registrar looks for the Subscription header field. For each value, it examines the event package and target resource. If the resource is in the domain of the registrar, and the resource has an authorization policy of "self", and the registrar allows registration coupled subscriptions for that event package, the registrar creates the dialog and a subscription. The 200 OK to the REGISTER contains an indication of whether the subscription was created, and if so, the remote tag needed to complete the dialog identifier.

The UAC will create a dialog and a subscription for each value of the

Subscription header field in the response. As there will be one of these per event package, the end result is a single dialog for each event package that the client wants to subscribe to. Dialogs are not shared across event packages. The dialog identifiers are obtained by copying the Call-ID and local tag from the REGISTER, with the remote tag from the Subscription header field value. Similarly, the registrar will create a subscription. The dialog identifiers and local sequence number are set in the same way. Its route set is taken from the Path header field from the registration [4].

At this point, a proper subscription is established at the UA and the registrar. The registrar can send a NOTIFY at any time. The initial NOTIFY normally sent upon receipt of a SUBSCRIBE is not required, as the REGISTER response serves that purpose. The subscriptions are all refreshed through registration refreshes. If the UAC omits an event and resource from a Subscription header field in its REGISTER, it means that the client wishes to unsubscribe. Similarly, if the 200 OK to the REGISTER omits that event package and resource, it means that the subscription was terminated. However, the client cannot ever send a SUBSCRIBE to refresh the subscription. Any such request is rejected with a 403.

It is important to note that there is a dialog properly established as part of this mechanism. The dialog is established by providing the dialog parameters through the registration, and then to make the dialog state part of the registration state. The dialog is then refreshed and maintained just like registration state. If a user has multiple user agents registered to the same AOR, multiple dialogs would be created. This means that the dialogs terminate on the registrar as well. In order for events to be delivered to the clients in NOTIFY messages, an event server generates a PUBLISH message when it wants to send an event to a user agent. The PUBLISH is routed to the registrar, where it examines the URI in the request URI. If the user is registered, it goes through each registered contact. If the registration of that contact had created a coupled subscription, the registrar checks if the registration-coupled subscriptions include the event package in the PUBLISH. If they do, the registrar copies the event data in the body of the PUBLISH into a NOTIFY, and sends it to the user agent.

As an additional mechanism, the event servers themselves can subscribe to the registration event package for all subscribers. WHenever a user registers, a notification would get delivered to the event server. It can then check which users are registered or not, and use this information to determine whether or not it wishes to send a PUBLISH. Alternatively, the reg-event notifications can contian all of the information on the registration-coupled subscriptions - their dialog identifiers, event packages, and so on.

This would allow the event server itself to "take over" the subscription, and take ownership of the dialog. In that case, it can send the NOTIFY directly, instead of sending a PUBLISH to the registrar. Indeed, the event server can make a decision on a subscriber-by-subscriber basis as to whether it wishes to own the dialogs or not.

3.2 User Agent Behavior

A user agent SHOULD be configured with a set of event packages that it wishes to couple with its registrations. For each such package, when the client performs its initial registration, it includes a Subscription header field value into its request. That value contains the address-of-record for the target of the subscription. This AOR MUST be one within the same domain as the domain of registration. Typically, it will be the same as the AOR for the user themselves. The UA includes any parameters it would otherwise include in the Event header field into the Subscription header field. The UA SHOULD include an Accept header field in the request, and include the content types the client supports for that event package. Otherwise, the registration is generated identically to a normal registration.

If the response to the REGISTER is a 200 OK, the client looks for the Subscription header field. If the header field is not present, the user agent knows that either this mechanism is not supported in the registar, or is supported, but not in use for any of the event packages requested by the client. In that case, the user agent SHOULD proceed with a normal subscription according to the specifics of the event packages the client is interested in.

If the 200 OK response to the REGISTER did contain a Subscription header field, the user agent goes through each value. It constructs a dialog by setting the Call-ID to the value in the REGISTER response, the local tag to the From tag the client placed in the REGISTER request, and the remote tag from the value of the Subscription header field. The local URI is set to the value in the From header field of the REGISTER request, and the remote URI to the value in the To header field of the REGISTER request. The local and remote CSeq are initially empty. Since the client never sends a request within the dialog, the local CSeq never needs to be populated. Similarly, the route set is empty. If the REGISTER request was sent over TLS, and the Request-URI was a sips URI, the "secure" flag for the dialog is set.

The dialog state persists for the duration of the registration of that contact. When the UA determines that the contact expires, the dialog state is destroyed. A UA can determine that a contact has

expired because it times out and is not refreshed, or because the client receives a registration event notification informing it that the contact has been terminated.

If the client had included a Subscription header field in the request for a particular event package, and the REGISTER response contained a Subscrption header field, but that package was not listed, it means that the registrar is either refusing a subscription-coupled registration for that event package, or that subscription failed for some reason. To determine the exact problem, the client SHOULD perform a regular, separate subscription to that event package.

At any point during the lifetime of the registration, the client may receive a NOTIFY on the dialog created by the registration. Processing of that NOTIFY happens as described in the relevant event package and according to the details of RFC 3265.

A registration refresh occurs identically to an initial registration. A client MUST include a Subscription header field value for each dialog it wishes to retain. If a client omits a Subscription header field value for a particular event package, the dialog associated with that event package is terminated upon receipt of a 200 OK to the REGISTER request.

If a client wishes to perform a subscription with event filters that need to be placed in the body of a request, the mechanism here cannot be used. Rather, the client should perform a normal subscription using SUBSCRIBE. An alternative would be to include the event filters as a body of the REGISTER request. Header field parameters could associated each MIME body with a particular event package. However, this introduces a lot of complexity for a corner case. As such, this document recommends just performing a regular subscription to handle these cases.

3.3 Registrar Behavior

3.3.1 REGISTER Processing

When a registrar receives a REGISTER request, it processes the registration normally per RFC 3261. If the result would otherwise have been a successful registration resulting in a 200 OK, the procedures defined here are followed.

The registrar checks for the presence of the Subscription header field in the REGISTER request. The processing that follows is performed for each value of this header field. Firstly, the registrar checks to see if it supports registration-coupled subscriptions for that particular event package. Performing them for

any particular event package is a matter of local policy. Typically, it would be allowed when an event server is present in the network which supports the capabilities defined here. If the registrar doesn't support registration-coupled subscriptions for that event package, it goes on to the next value of the Subscription header field. Otherwise, processing continues.

Next, the registrar validates that the resource in the header field value is a valid resource within the domain of the registrar. If it is, processing continues. Otherwise, the registrar goes on to the next value of the Subscription header field. Next, it checks whether or not the UAC is authorized to subscribe to the resource. The means by which authorization occurs is outside the scope of this specification. Typically, registration-coupled subscriptions are performed with subscriptions where the authorization policy is such that a user is allowed to subscribe to themselves, and no others. This authorization policy, called "self", is readily provisioned on the registrar, and would not require complex interactions with other event servers. If the registrar cannot determine authorization, or if the subscription is not authorized, the registrar goes on to the next value of the Subscription header field. Otherwise, processing continues.

At this point, the subscription has been authorized. The registrar stores the event header field parameters in the Subscription header field value as part of the state associated with the registered contact. These parameters are carried as a quoted string in the Subscription header field, so that they are readily separable from the Subscription header field parameters. It also stores the event package. The registrar chooses a tag that will serve as the remote tag of the dialog, according to the procedures of RFC 3261. This tag is also stored as part of the state associated with the registered contact. The Call-ID and From tag from the REGISTER request would have already been stored as part of normal registration processing, as would the Path header field value. The registrar also stores the From header field of the REGISTER message.

In the 200 OK to the REGISTER request, the registrar includes the Subscription header field. Each value contains the event package name for each registration-coupled subscription that was created, along with the tag that completes the dialog. The AOR SHOULD NOT be included.

3.3.2 PUBLISH Processing

This specification allows a registrar to act as an event server for registration-coupled subscriptions. When the registrar receives a PUBLISH message for a particular address-of-record, it checks that

the PUBLISH has arrived from an event server that is authorized to publish events for the subscriber. Typically, this is done based on the maintentance of a TLS connection between the registrar and the event server, used to identify the source of the messages to the registrar. The registrar would typically authorize PUBLISH messages for a specific event package only if they came from a specific event server.

Once the sender of the PUBLISH is authorized, the registrar performs a registration query for the AOR in the Request-URI of the PUBLISH message. It checks to see if there are any contacts registered for that AOR that have registration-coupled subscriptions for that event package. For each contact it finds, the registrar constructs a NOTIFY message. The Call-ID of this NOTIFY is taken from the stored state associated with the registration. The From header field URI is set to the AOR of the user. The To header field URI is set to the value in the From header field of the most recent REGISTER message. The tag in the From header field is populated with the tag associated with the registration. The tag in the To header field is populated with the tag stored with the Contact. The Event header field of the NOTIFY is set to the event header field stored with the Contact. The body of the NOTIFY is taken from the body of the PUBLISH. The remainder of the NOTIFY is consructed as per RFC 3261, and then sent as a mid-dialog request.

The registrar then generates a 200 OK to the PUBLISH request. If the registrar found no matching registration-coupled subscriptions for the PUBLISH, it generates a 403 response to the PUBLISH request. This informs the event server that its event was not delivered.

3.4 Event Server Behavior

It is assumed that event servers learn about events for a particular package for a particular subscriber through any number of means. These can include non-SIP mechanisms, SIP subscriptions to a resource, and so on. However, they cannot include a SIP PUBLISH message sent to the AOR of the subscriber; those PUBLISH messages are routed to the registrar according to this specification.

An event server MAY act as the dialog owner, or MAY leave that responsibility to the registrar. However, it MUST NOT do both for the same subscriber within the duration of a registration from that subscriber. To act as a dialog owner, the event server subscribes to the registration event package. It MAY subscribe to this event package for each subcscriber individually, or it MAY subscribe to a resource that represents all subscribers or a group of users at the registrar (for example, sip:all-users@example.com). The latter is preferable since it avoids the need for per-user subscription

maintenance at the event server.

The notifications of the dialog event package will contain information on each registration-coupled subscripton for a subscriber. If the event server is acting as a dialog owner, it MUST store this information. Effectively, the reg-event notification creates the dialog state and the event subscription at the event server. When the event server wishes to send an event, it creates a NOTIFY using the dialog state and sends it, per RFC 3265 and RFC 3261 procedures. These NOTIFY messages won't even traverse the registrar.

If the event server is not acting as a dialog owner, when it wishes to send a notification, it sends a PUBLISH message. The request-URI of the PUBLISH is set to the AOR of the subscriber for whom a notification is to be delivered. The content of the PUBLISH contains the event state that is to be delivered to the watcher. The Event header field is populated with the value of the event package for which the notifications are intended. This PUBLISH message is sent, and will be routed to the registrar. The processing above will result in a NOTIFY being sent to each registered contact for that AOR.

The choice of whether to act as dialog owner or not depends on several factors. When the event server leaves dialog ownership to the registrar, it alleviates the need for the event server to maintain any kind of per-subscriber state. However, it imposes additional work on the registrar to perform the registration queries and construction of NOTIFY messages. Thus, this mode is useful for very infrequent events, such as a request to update a configuration profile in the configuration event package. Dialog ownership makes more sense for more frequent events. Also, since the registrar doesnt know the actual event state, it cannot send an initial NOTIFY with the current state when the dialog is first created. It relies on the event server to do that. As a result, if an event package requires state to be delivered as part of a NOTIFY generated when the subscription is created, the event server needs to maintain ownership of the dialog, or the hybrid model below needs to be used.

A hybrid model is also possible. An event server can receive reg-event notifications, but not store dialog state. When it sees that the user has registerd or unregistered, it can send a PUBLISH message. This is useful for infrequent notifications that need to be triggered on registration. The hybrid model also allows the event server to generate a PUBLISH when a client first registers, that contains the current value of the event state. This will cause the registrar to send a NOTIFY message with the current state. This is useful for event packages where it is desireable to send event state as part of the initial NOTIFY.

The hybrid model is particularly attractive, since it alleviates the need for the event server to maintain any kind of dialog state or per-subscriber subscription state, and yet it allows for the full features of a traditional event subscription.

3.5 Subscription Header Field

The grammar for the Subscription header field is:

```
Subscription    =  "Subscription" HCOLON (sub-param *(COMMA
                   sub-param))
sub-param       =  event-type *(SEMI sub-param)
sub-event-param =  sub-aor / sub-event-param / tag-param / generic-param
sub-aor         =  "aor" EQUAL (SIP-URI / SIPS-URI)
sub-event-param =  "e-param" EQUAL quoted-string
```

Figure 3 and Figure 4 are an extension of Tables 2 and 3 in RFC 3261 [1] for the Subscription header field. The column "INF" is for the INFO method [5], "PRA" is for the PRACK method [6], "UPD" is for the UPDATE method [7], "SUB" is for the SUBSCRIBE method [2], "NOT" is for the NOTIFY method [2], "MSG" is for the MESSAGE method [8], "PUB" is for the PUBLISH method [9], and "REF" is for the REFER method [10].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	REF
Subscription	R	-	-	-	-	-	-	o	-
Subscription	2xx	-	-	-	-	-	-	o	-

Figure 3: Subscription header field

Header field	where	proxy	PRA	UPD	SUB	NOT	INF	MSG	PUB
Subscription	R	-	-	-	-	-	-	-	-
Subscription	2xx	-	-	-	-	-	-	-	-

Figure 4: Subscription header field

3.6 Examples

3.6.1 Registrar has Dialog Ownership

In this example, the registrar holds ownership of the dialog. The event server is a message waiting indicator server that publishes MWI events.

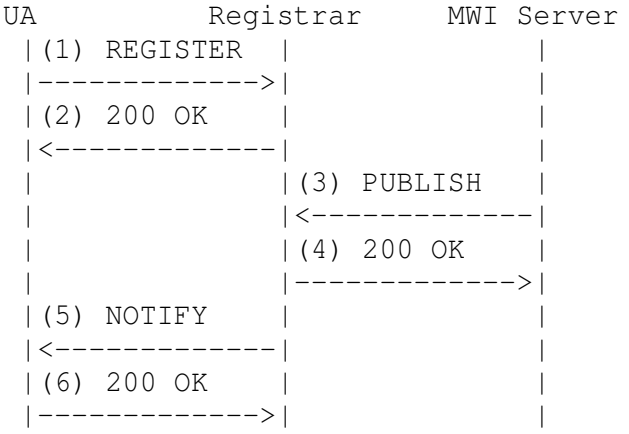


Figure 5: Registrar Owned Dialogs

The REGISTER message (1) would look like:

```
REGISTER sip:example.com SIP/2.0
To: sip:joe@example.com
From: sip:joe@example.com;tag=asd9887g
Subscription: message-summary;aor=sip:joe@example.com
Expires: 3600
Via: SIP/2.0/UDP client.biloxi.example.com;branch=z9hG4bKnashds7
Max-Forwards: 70
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 REGISTER
Content-Length: 0
Contact: sip:client.biloxi.example.com
```

The 200 OK to the REGISTER indicates successful creation of the dialog:

```
SIP/2.0 200 OK
To: sip:joe@example.com;tag=99j9jj
From: sip:joe@example.com;tag=asd9887g
Subscription: message-summary;tag=ghghghg
Expires: 3600
Via: SIP/2.0/UDP client.biloxi.example.com;branch=z9hG4bKnashds7
Max-Forwards: 70
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 REGISTER
Content-Length: 0
```

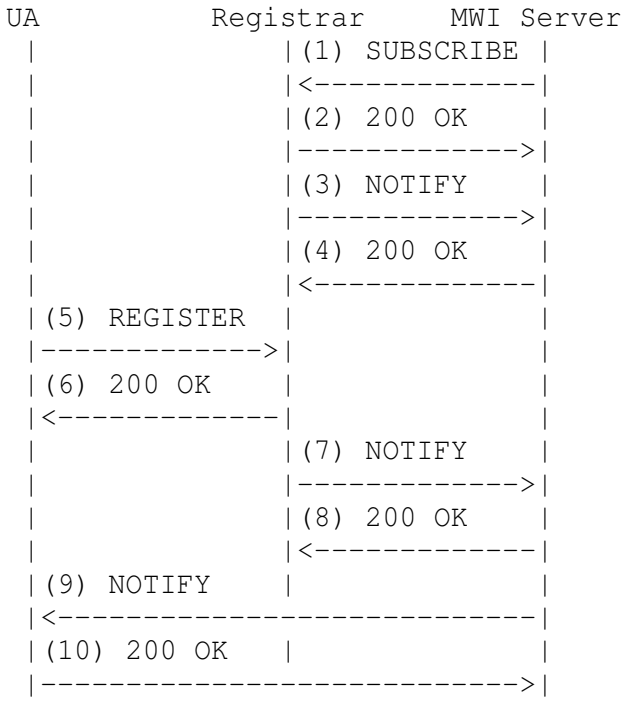
The PUBLISH from the event server comes when a new message arrives:

```
PUBLISH sip:joe@example.com SIP/2.0
To: sip:joe@example.com
From: sip:mwi-server@example.com
Event: message-summary
Via: SIP/2.0/UDP mwi.example.com;branch=z9hG4bKnashas--d9
Call-ID: 3k9FpLxhg88asd7m8tn@mwi.example.com
CSeq: 1 PUBLISH
Content-Type: application/simple-message-summary
Content-Length: ---

Messages-Waiting: yes
Message-Account: sip:joe@mwi.example.com
Voice-Message: 2/8 (0/2)
```

This results in a notification from the registrar:

```
NOTIFY sip:client.biloxi.example.com SIP/2.0
To: sip:joe@example.com;tag=asd9887g
From: sip:joe@example.com;tag=ghghghg
Event: message-summary
Via: SIP/2.0/UDP reg.example.com;branch=z9hG4bKnashas--d10
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 NOTIFY
Content-Type: application/simple-message-summary
Content-Length: ---

Messages-Waiting: yes
Message-Account: sip:joe@mwi.example.com
Voice-Message: 2/8 (0/2)
```

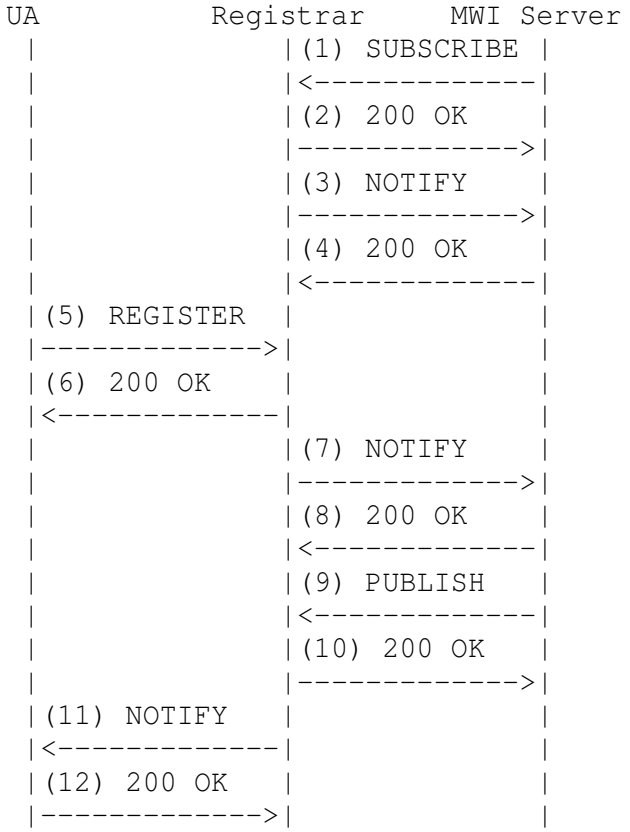
3.6.2 Event Server Owned Dialog



When the message waiting server starts up, it subscribes to the registration event package at the registrar (message 1). The request URI identifies all users in the domain. This generates a 200 OK (message 2), followed by a NOTIFY (message 3). This NOTIFY doesn't contain any event state (there is too much), but it confirms the subscription.

At some point later, the UA in question registers. The registration sequence (messages 5/6) are as above. This causes a reg-event NOTIFY to be sent to the mwi server (message 7). This tells the server about the creation of a new contact, and also tells it that a MWI registration-coupled subscription was created. It provides the dialog identifiers to the MWI server. Next, the MWI server generates a NOTIFY to tell the client about the event state (9).

3.6.3 Hybrid Model



When the message waiting server starts up, it subscribes to the registration event package at the registrar (message 1). The request URI identifies all users in the domain. This generates a 200 OK (message 2), followed by a NOTIFY (message 3). This NOTIFY doesn't contain any event state (there is too much), but it confirms the subscription.

At some point later, the UA in question registers. The registration sequence (messages 5/6) are as above. This causes a reg-event NOTIFY to be sent to the mwi server (message 7). This tells the server about the creation of a new contact, and also tells it that a MWI registration-coupled subscription was created. It provides the dialog identifiers to the MWI server. However, instead of sending the NOTIFY, the MWI server discards the dialog information. It sends a PUBLISH request (message 9) identically to the case where the registrar owns the dialog. This causes the registrar to send the notification (message 11).

4. References

4.1 Normative References

[1] 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.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration", RFC 3608, October 2003.

[4] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[5] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[6] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

[7] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

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

[9] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

[10] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

4.2 Informative References

[11] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)", RFC 3842, August 2004.

[12] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[13] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[14] Roach, A., Rosenberg, J., and B. Campbell, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", draft-ietf-simple-event-list-07 (work in progress), January 2005.

[15] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[16] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", RFC 3857, August 2004.

[17] Rosenberg, J., "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-06 (work in progress), April 2005.

[18] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-06 (work in progress), February 2005.

Author's Address

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP
Internet-Draft
Expires: January 7, 2006

H. Tschofenig
Siemens
J. Peterson
NeuStar, Inc.
J. Polk
Cisco
D. Sicker
CU Boulder
M. Tegnander
LYIT
July 6, 2005

Using SAML for SIP
draft-tschofenig-sip-saml-03.txt

Status of this Memo

Copyright Notice

Abstract

This document describes how to use the Security Assertion Markup

Language (SAML) to offer trait-based authorization. As such, it provides an alternative to existing authorization mechanisms for SIP.

Table of Contents

1. Introduction

This document proposes a method for using the Security Assertion Markup Language (SAML) in collaboration with SIP to accommodate richer authorization mechanisms and enable trait- based authorization where you are authenticated using roles or traits instead of identity. A motivation for trait based authorization and some scenarios are presented in [I-D.ietf-sipping-trait-authz].

Security Assertion Markup Language (SAML) [I-D.saml-tech-overview-1.1-03] is an XML extension for security information exchange that is being developed by OASIS. SAML is a XML-based framework for creating and exchanging security information.

To provide trait-based authorization a few solutions are possible: authorization certificates, SPKI or extensions to the authenticated identity body [I-D.ietf-sip-authid-body]. The authors selected SAML due to the amount of work done in the area of SAML which provides some assurance that this technology is mature enough.

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].

The SIP entity 'Authentication Service' was introduced with [I-D.ietf-sip-identity]. We reuse this term to refer to an entity that authenticates and authorizes a user and creates an assertion. This entity is the equivalent of the asserting party in the SAML terminology.

For terminology related to SAML the reader is referred to [I-D.saml-tech-overview-1.1-03].

3. Goals and Non-Goals

This document tries to accomplish the following goals:

- This document defines how SAML assertions are carried in the SIP. As such, the usage of SAML assertions within SIP can be seen as a SAML profile.

- The requirements and scenarios defined in [I-D.ietf-sipping-trait-authz] are compared to the solution described in this document by utilizing SAML assertions.

The following issues are outside the scope of this document:

- The configuration of the Authentication Service in order to attach certain assertions is outside the scope of this specification and might depend on the environment where SIP is used. To avoid restricting the functionality of SIP either as an in-band or an out-of-band mechanism, it can be defined to trigger the inclusion of SAML assertions. SAML itself provides mechanisms for this purpose.

- The attributes stored in assertions are, for example, roles, membership to a certain organization, specific access rights or information about the authentication. A definition of most of these attributes is application dependent and not defined in this document. The SAML specification itself provides a number of common attributes and provides extension points for future enhancements. A brief overview of the available attributes within an assertion is given in Section 4.1.

- SIP is not used as a request/response protocol between the Relying Party and the Asserting Party to fetch an assertion based on a received artifact.

4. SAML Introduction

In SAML there are three main entities: the user, the asserting party and the relying party. A user requests an assertions and receives them after a successful authentication and authorization protocol execution. The asserting party provides assurance that a particular user has been given proper authorization. The relying party has to trust the asserting party with regard to the provided information and then decides whether or not to accept the assertions provided, giving different levels of privileges.

The components of SAML are:

- Assertions/Artifact
- Request/Response protocols
- Bindings
- Profiles

We describe each in turn below

4.1 Assertions

An assertion is a package of information including authentication statements, attribute statements and authorization decision statements. All of statements do not have to be present, but at least one does. An assertion contains several elements:

Issuing information:

Who issued the assertion, when was it issued and the assertion identifier.

Subject information:

The name of the subject, the security domain and optional subject information, like public key.

Conditions under which the assertion is valid:

Special kind of conditions like assertion validity period, audience restriction and target restriction.

Additional advice:

Explaining how the assertion was made, for example.

In an authentication statement, an issuing authority asserts that a certain subject was authenticated by certain means at a certain time.

In an attribute statement, an issuing authority asserts that a certain subject is associated with certain attributes which has certain values. For example, user jon@cs.example.com is associated with the attribute 'Department', which has the value 'Computer Science'.

In an authorization decision statement, a certain subject with a certain access type to a certain resource has given certain evidence that the identity is correct. Based on this, the relying party then makes the decision on giving access or not. The subject could be a human or a program, the resource could be a webpage or a web service, for example.

4.2 Artifact

The artifact used in the Browser/Artifact profile, is a base-64 encoded string that is 40 bytes long. 20 bytes consists of the typecode, which is the source id. The remaining 20 bytes consists of a random number that servers use to look up an assertion. The source server stores the assertion temporarily. The destination server receives the artifact and pulls the assertion from the source site. The purpose of the artifact is to act as a token that references an assertion for the subject who holds the artifact.

4.3 Request/Response Protocol

SAML defines a request/response protocol for obtaining assertions. The request asks for an assertion or makes queries for authentication, attribute and authorization decisions. The response carries back the requested assertion.

4.4 Bindings

The bindings in SAML maps between the SAML protocol and a transport and messaging protocol. With SAML Version 1.1 there is only one binding specified, which is SAML embedded in SOAP-over-HTTP. In a binding, a transport and messaging protocol is used only for transporting the request/response mechanism.

4.5 Profiles

When using a profile, SAML is used to provide assertions about a resource in the body of the message itself. In Version 1.1 of SAML, there are two profiles specified, the Browser/Artifact profile and the Browser/POST profile. The Browser/Artifact profile represents a "pull" model, where a special reference to the assertion called an artifact, is sent to the relying party from the asserting party. The artifact is then used to "pull" the assertion from the asserting party. The Browser/POST profile represents a "push" model, where an assertion is posted (using the HTTP POST command) directly to the relying party. These two models are described in Figure 1 and Figure 2.

5. Assertion Handling Models

As mentioned in Section 4.5, two main models can be used in SAML and therefore also with the SIP-SAML extension defined in this document: The Push and the Pull model.

In the Pull model the end host requests an assertion from the Asserting Party and receives, after successful authentication and authorization, an artifact. The artifact is a special form of an assertion. This artifact can be compared with the call-by reference approach where a reference to the assertion is stored at the Asserting Party and can later be dereferenced into the real assertion on a request by a replying party. The Relying Party later fetches the SAML assertion after receiving a request by the user which includes the artifact. For communicating the SAML request and response messages, a separate message exchange is needed with a protocol such as SOAP or HTTP. This is outside the scope of this document.

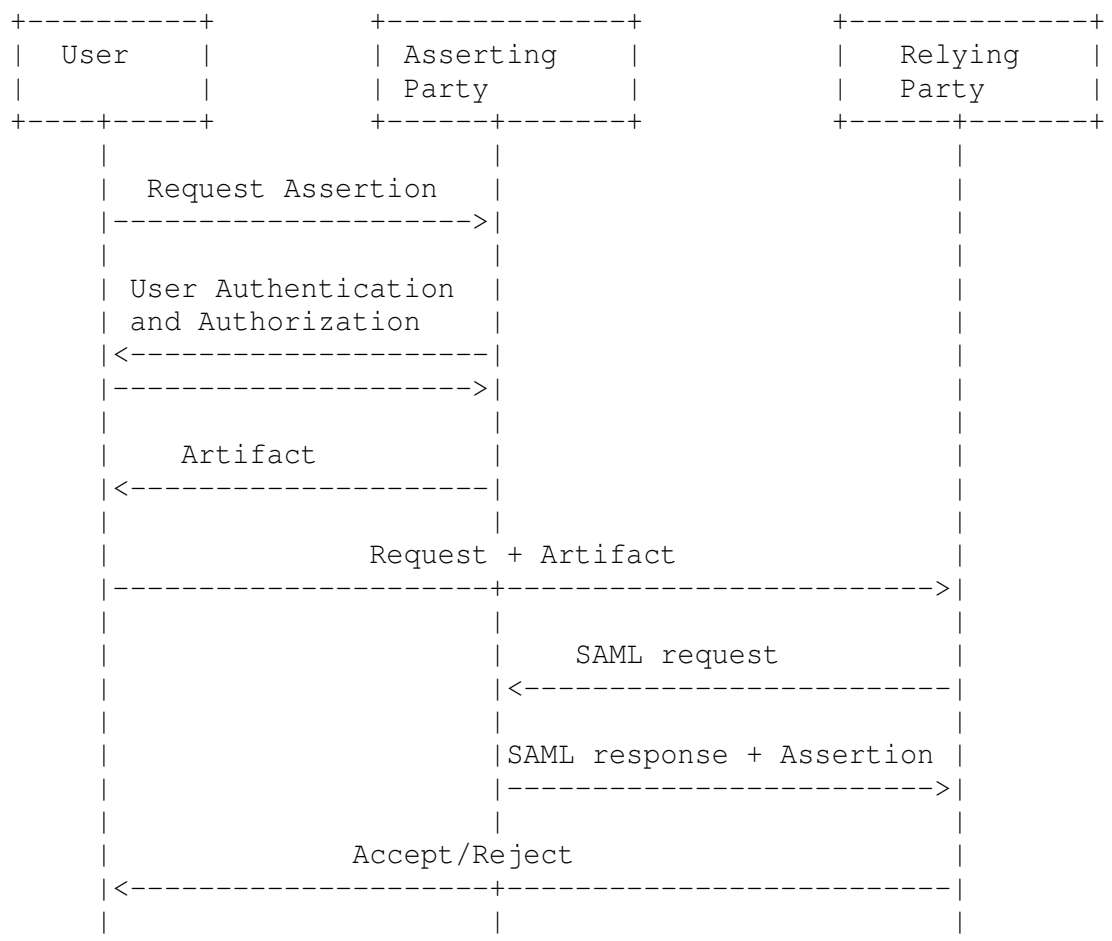


Figure 1: SAML Pull model

With the Push model, the user requests an assertion from the Asserting Party. The user can also trigger the Asserting Party to attach an assertion to the request. The assertion, which is added to the service request, can be verified by the Relying Party without additional protocol interactions with the Asserting Party. The assertion therefore contains enough information to authorize the service request. This model realizes a call-by value style of communication.

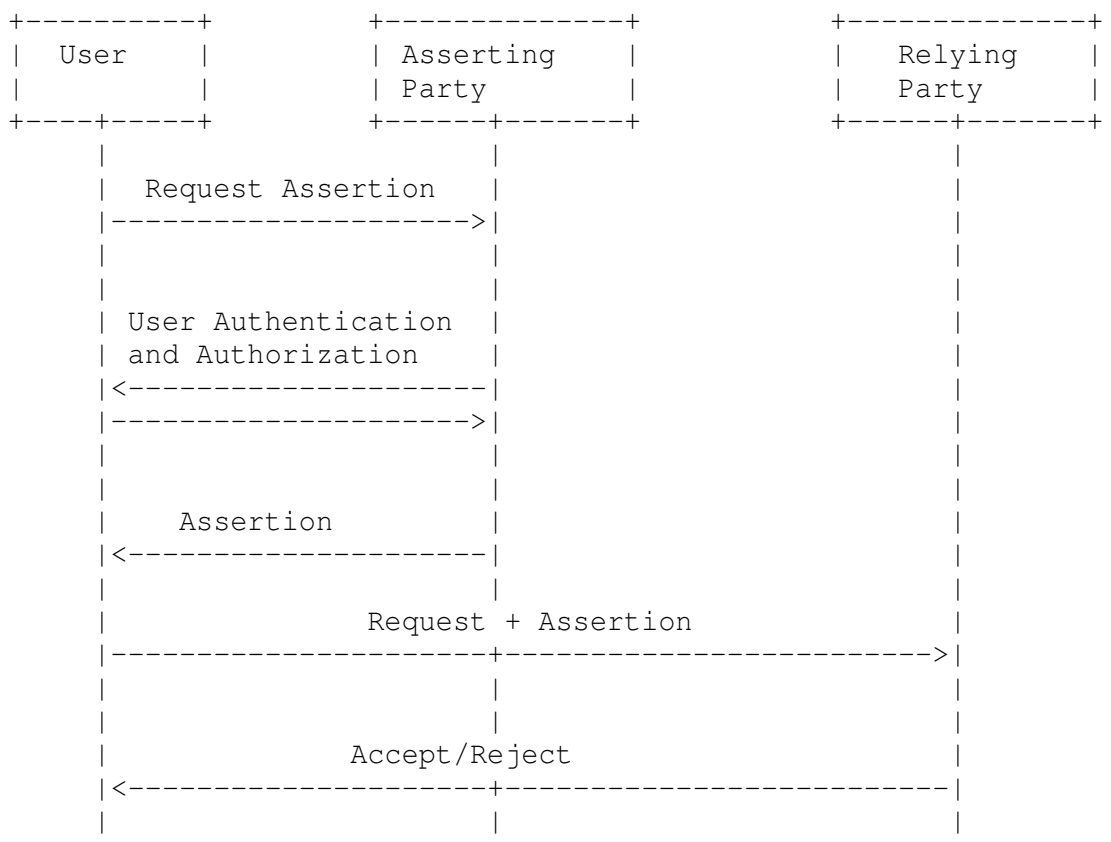


Figure 2: SAML Push model

The usage of SAML in HTTP-based environments and in SIP is, however, affected by some architectural differences.

The function of the entities in the Push and the Pull model are shown in Figure 1 and in Figure 2.

The user has to request an assertion at the Asserting Party and both entities mutually authenticate each other. The requested assertion is sent to the user in a confidential manner to prevent eavesdroppers from obtaining this assertion. The Relying Party trusts the Asserting Party. It is assumed that the accessed resource is located at the Relying Party and that this entity does not become malicious or act on behalf of the user to impersonate him or her to other parties with regard to access to his resources. To prevent some degree of misuse, attributes in the assertion limit its applicability for certain applications, servers or time frame.

Signaling in SIP can, however, involve a number of entities in more complex scenarios. As an example, the scenario addressed in [I-D.ietf-sip-identity] aims to replace end-to-end authentication via

S/MIME by a "mediated authentication architecture". The end hosts only need to be able to verify assertions signed by an Authentication Service which guarantees that the sender was authenticated.

Directly applying SAML to such a scenario, however, causes a problem: a SIP proxy, which securely receives a SAML assertion (such as confidentially protected to prevent eavesdroppers between the SIP UA and the SIP proxy to learn the assertion), can store this assertion to impersonate the user in future requests towards other SIP end users. The fact that multiple parties see the assertion along the path (i.e., SIP proxies) make the situation worse. The assertion might include some attributes which restrict its usage (such as recipient(s), unique identifier for the message or a time-based constraint) but they cannot fully prevent impersonation.

This problem appears if SAML assertions are not bound to a particular protocol run. Binding the assertion to a particular protocol session is not useful if the property of single-sign on is required.

To provide referential integrity, a solution as mentioned in [I-D.ietf-sip-authid-body] can be used. which allows a party in a SIP transaction to cryptographically sign the headers that assert the identity of the originator of a message, and provide some other headers necessary for reference integrity. An authenticated identity body (AIB) is a MIME body of type 'message/sipfrag'. This MIME body has a Content-Disposition type of 'aib'. The MIME body is optional. The header fields From, Contact, Date and Call-ID must be used for providing identity. Contact and Date header fields are required for providing reference integrity. AIBs may contain other headers that help to uniquely identify the transaction or that provides related reference integrity.

The requirements for a non-INVITE AIB is very much the same as for an INVITE: From, Call-ID, Date and Contact header fields are required. The same that goes for requests also goes for responses with some small differences. The From header field of the AIB in the response to an INVITE must correspond to the address-of-record of the responder and not the From header field in the received request. The To header field of the request must not be included. A new Date header field has to be generated for the response while the Call-ID and CSeq are copied from the request.

Following is an example of the format of an AIB for an INVITE:

```
Content-Type: message/sipfrag
Content-Disposition: aib; handling=optional

From: Alice <sip:alice@example.com>
To: Bob <sip:bob@example2.com>
Contact: <sip:alice@pc33.example.com>
Date: Thu, 26 Aug 2004 13:51:34 GMT
Call-ID: b76m5l94s90835
CSeq: 435431 INVITE
```

Figure 3: AIB Format for an INVITE

The same concept is applicable to this document as well with regard to reference integrity. For a further discussion on this topic see Section 14 and [I-D.peterson-message-identity].

6. Scenarios

This section shows message flows based on scenarios in [I-D.ietf-sipping-trait-authz] enriched with a SAML based solution. Section 6.1 provides an example of enhanced network asserted identities and Section 6.2 shows a SIP conferencing scenario with role-based access control using SAML. A future version of this document will cover more scenarios from [I-D.ietf-sipping-trait-authz].

6.1 Network Asserted Identities

Figure 4 shows an enhanced network asserted identity scenario based on [I-D.ietf-sip-identity] which again utilizes extensions proposed with [I-D.ietf-sip-authid-body]. The enhancement is based on the attributes asserted by the Authentication Service.

Figure 4 shows three entities, Alice@example.com, AS@example.com and Bob@example2.com. If Alice wants to communicate with Bob, she sends a SIP INVITE to her preferred AS. Depending on the chosen SIP security mechanism either digest authentication, S/MIME or Transport Layer Security is used to provide the AS with a strong assurance about the identity of Alice. During this step authorization attributes for inclusion into the SAML assertion can be selected.

After Alice is authenticated and authorized, a SAML assertion is attached to the SIP message. The Authentication Service can be configured to attach a number of assertions, not limited to the authenticated identity.

To bind the SAML assertion to a specific SIP session, it is necessary for the AS to compute a hash of some fields of the message. A list of the fields to hash is described in [I-D.ietf-sip-identity] and particularly in [I-D.ietf-sip-authid-body]. The hash is digitally signed and inserted into the SAML assertion and placed into the SAML header. The SAML header also contains information about the entity which created the digital signature. Upon reception of the message, Bob verifies the signature of the SAML assertion and verifies the binding to the SIP message in order to prevent cut-and-paste attacks. The provided SAML assertion can then be used to assist during the authorization procedure. If Bob does not understand the extension defined in this document, he silently ignores it. When the 200 OK message arrives at Bob's AS, the same procedure as when Alice sent her INVITE to her AS can be performed, if desired. This exchange is not shown in Figure 4.

Note that this scenario does not imply that the SAML assertions are solely used by SIP UAs. Assertions can also be helpful for SIP

proxies or B2B UAs. Additionally, a push model is shown in this section but it is reasonable to use a pull as well. For simplicity reasons a push model should be prefered since an additional message exchange between the Authentication Service and the UA can be omitted.

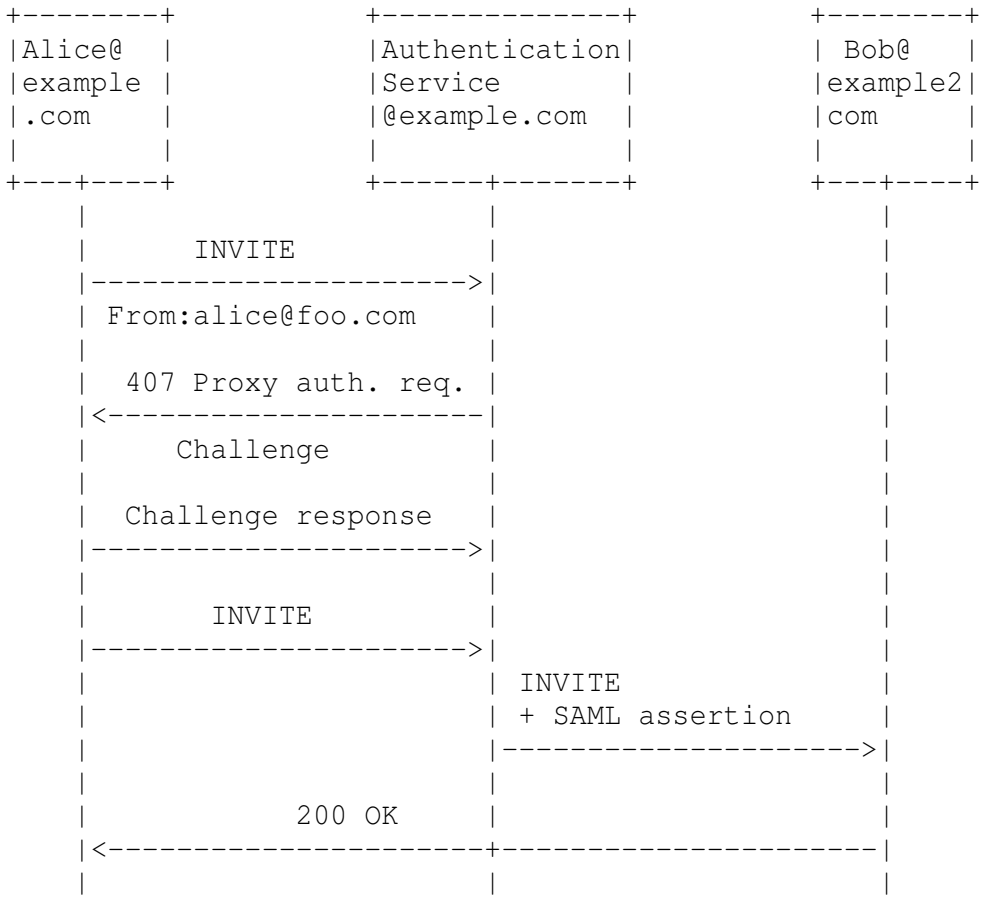


Figure 4: Network Asserted Identities

A variation of the scenario presented in Figure 4 is given in Figure 5 where an end host (Alice@example.com) obtains an assertion from its SIP proxy server which acts as an Authentication Service. This assertion is then attached by the end host to the outgoing INVITE message. Unlike in case of an artifact, Bob@example.com does not need to contact the Proxy Server.

An example of this scenario could be to preempt a lower priority call if enough assurance for doing so is presented in the attached SAML assertion. This would also mean that there is a priority value included in the INVITE (for example in the Resource-Priority Header).

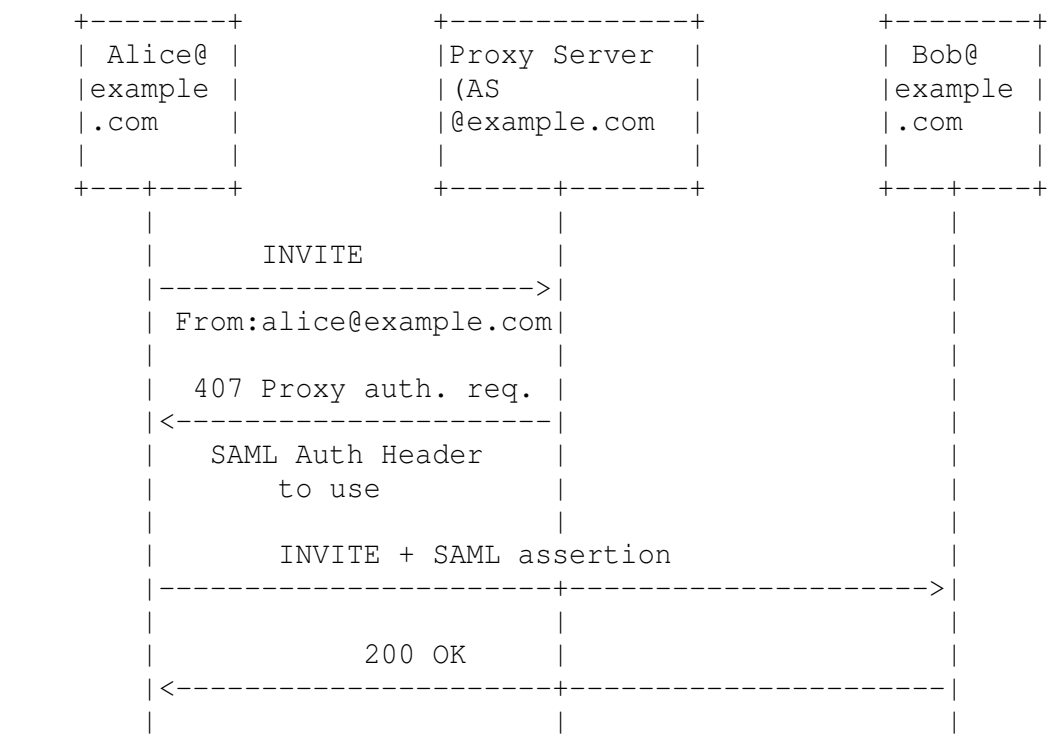


Figure 5: End host attaching SAML Assertion

Note that Bob and Alice do not need to be in the same administrative domain. It is feasible that Bob is in a domain that is federated with Alice's domain.

The assertion obtained by Alice@example.com needs to be associated with a particular SIP messaging session. How to achieve this binding is for further consideration.

6.2 SIP Conferencing

This section is meant to raise some discussions about the usage of SAML in the domain of conferencing. A user who routes its SIP message through the Authentication Service (Asserting Party) towards a conferencing server may want SAML assertions to be included. The following properties could be provided by this procedure:

- The user identity can be replaced to allow the user to be anonymous with regard to the Focus
- The user identity could be asserted to the Focus
- The SAML assertion could provide additional information such as group membership (belongs to the students, staff, faculty group of

university X). This could, for non-identity-based authorization systems, imply certain rights.

The corresponding SIP message flow (in high level detail) could have the following shape:

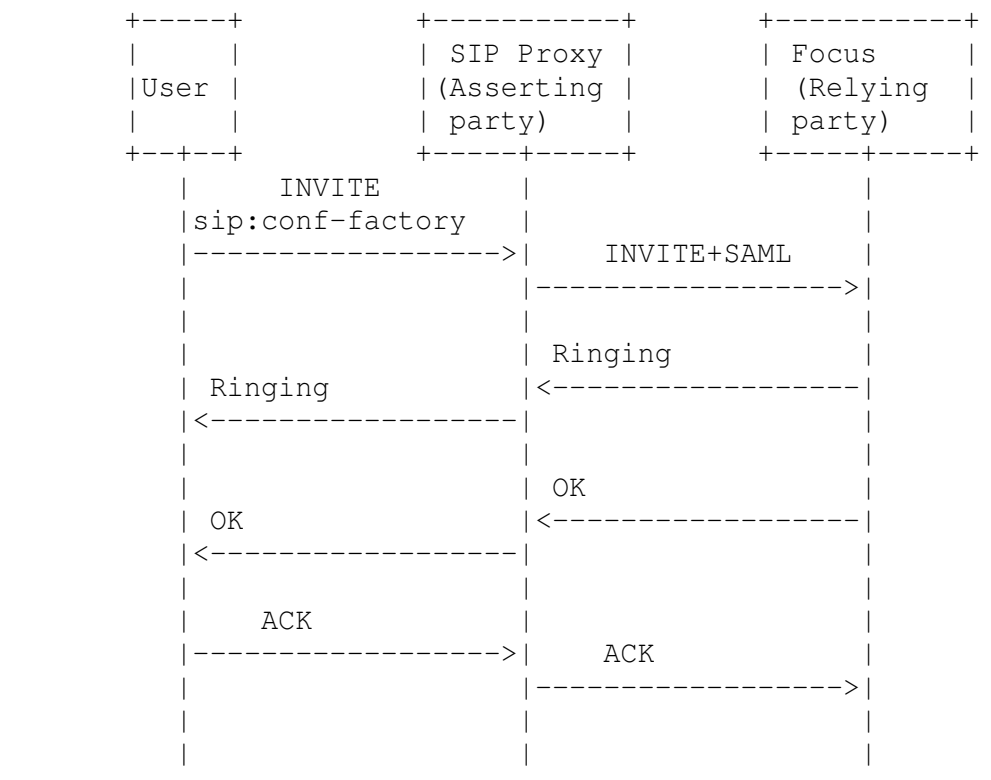


Figure 6: SIP Conferencing and SAML

6.3 PSTN-to-SIP Phone Call

Alice, using a phone connected to the PSTN, wants to make a call to Bob, which resides in a SIP network. Her call is switched through the PSTN by means of PSTN signaling (outside the scope of this document) to the PSTN/SIP gateway. At the gateway, the call is converted from SS7 signaling to SIP signaling. Since Alice was previously 'authenticated' through PSTN signaling mechanisms, the gateway can create an assertion based on signaling information from Alice and digitally sign it with his private key. Alice's call is forwarded from the SIP/PSTN gateway to Bob's phone. Bob can certify that the identity of Alice is correct by examining the SAML assertion.

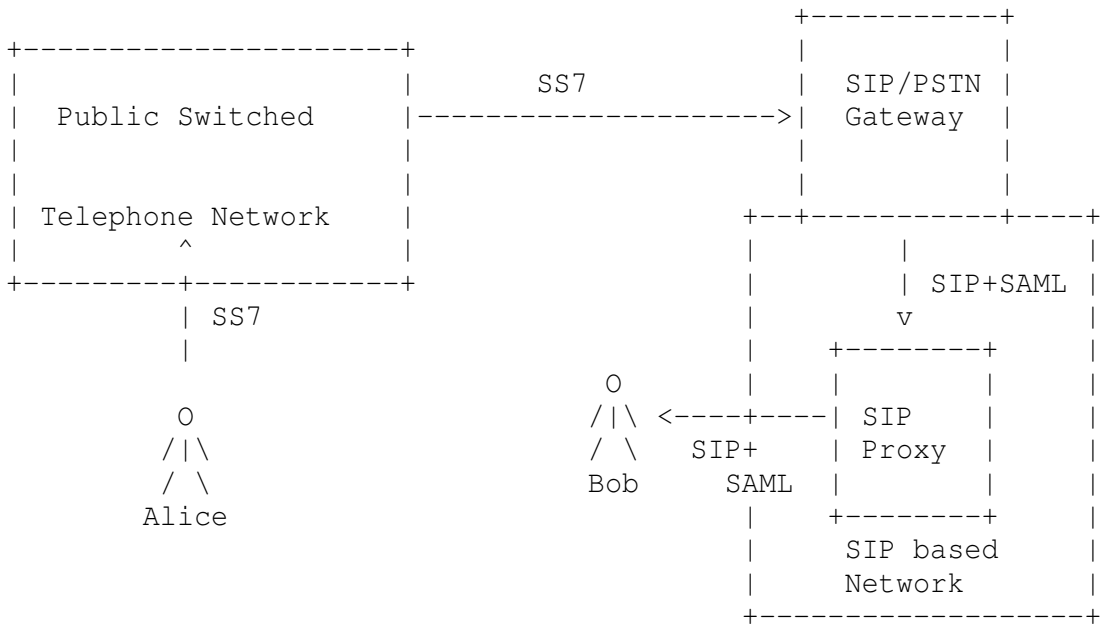


Figure 7: PSTN to SIP call

6.4 Compensation using SIP and SAML

This section briefly elaborates a scenario where SAML is used in SIP to realize compensation functionality as described in [I-D.jennings-sipping-pay]

Section 1 of [I-D.jennings-sipping-pay] shows a message exchange which is used by a number of payment protocols and hence can also be used by a SAML specified protocol. To avoid repetition in this document a second alternative is provided in Figure 8. Alice initiates a communication with an Authentication Service which acts as a financial institution. Note that Alice does not necessarily need to use SIP for communication with the Authentication Service. Instead, it might be possible to use HTTP or other protocols which offer the necessary user credential or offer additional information (such as a web page). After a successful authentication and authorization Alice obtains an assertion (or an artifact) which might contain payment relevant information. For a later service access, Alice contacts the merchant Bob with the assertion. Bob verifies the assertion and, it might want to contact the Authentication Service for a credit check. A financial settlement between the merchant Bob and the Trusted Third Party is assumed. Depending on the type of service, a one-time payment with immediate amount deduction may take place (e.g., in case of a prepaid account) or the amount is captured as a batch transaction.

The aspect of lightweight protocol execution is provided by

- The alternative usage of an artifact which leads to a lower bandwidth consumption.

- The ability to use a single assertion for multiple service access protocol executions, if desired.

- SAML, furthermore allows a cryptographic key to be bound to an assertion. A high degree of flexibility is provided with regard to the possible credentials. For example, it might not be necessary to use public key cryptography with every service access. This might be useful if the cost of public key cryptographic is too expensive for a cheap service or when devices have performance limitations. In this case, it might be useful to rely on symmetric cryptographic, such as hash chains.

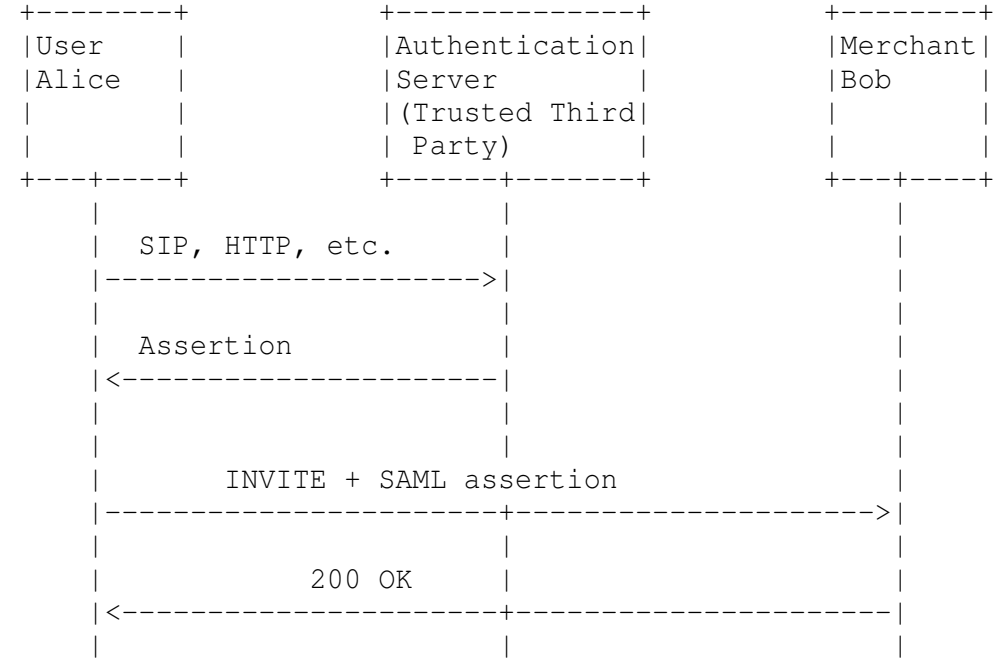


Figure 8: Message flow for SIP payment

The main difference between the above-described mechanism and the one described in Section 1 of [I-D.jennings-sipping-pay] is the degree of user involvement and the type of protocol interaction. In both cases it is possible to provide an indication to the user about the costs of a service access. In fact, the assertion might specify these type of constraints directly or indirectly with the help of recurring service requests/responses.

7. SIP-SAML Extension

To carry SAML assertions and artifacts two mechanisms are defined:

- The SIP header may either carry an Artifcat or a CID URI [RFC2392] pointing to an assertion in the SIP body. The name of this new SIP header is SAML-Payload. A SAML artifact consists of a TypeCode, SourceID and an AssertionHandle. It is thereby assumed that the Relying Party will maintain a table of sourceID values as well as URLs for Asserting Parties to contact. This information is communicated out-of-band. This document also allows the Asserting Party to add a URL to point to the assertion to prevent this level of indirection.

- The SIP body may carry one or more SAML assertions. The MIME type of this SAML assertion is defined in [I-D.hodges-saml-mediatype].

A SIP user agent may add an assertion to the body of a SIP message or may add a reference to the assertion into the SIP header. SIP proxies MUST NOT add the assertion to the body. The SIP header MUST be used instead when an assertion has to be added.

A SAML assertion SHOULD be protected and when added by a SIP entity then S/MIME MUST be used rather than XML digital signatures.

To bind a SAML assertion to a SIP message a few selected SIP message fields are input to a hash function. The digest-string, defined in Section 10 of [I-D.ietf-sip-identity], is included into the conditions extension point of the SAML assertion. Details are for further study.

8. Example

This is an example of a SAML assertion and how it is structured in XML.

```
<saml:Assertion
 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
 MajorVersion="1"
 MinorVersion="1"
 AssertionID="P1YaAz/tP6U/fsw/xA+jax5TPxQ="
 Issuer="www.example.com"
 IssueInstant="2004-06-28T17:15:32.753Z">
       <saml:Conditions NotBefore="2004-06-28T17:10:32.753Z"
         NotOnOrAfter="2004-06-28T17:20:32.753Z" />
       <saml:AuthenticationStatement
     AuthenticationMethod="urn:ietf:rfc:3075"
     AuthenticationInstant="2004-06-28T17:15:12.706Z">
          <saml:Subject>
                  <saml:NameIdentifier>
                    NameQualifier=alice@example.com
                    Format="urn:oasis:names:tc:SAML:1.1:nameid-
                     format:emailAddress">uid=alice
                  </saml:NameIdentifier>
                  <saml:SubjectConfirmation>
                              <saml:ConfirmationMethod>
                                urn:oasis:names:tc:SAML:1.0:
                                 cm:SIP-artifact-01
                              </saml:ConfirmationMethod>
                  </saml:SubjectConfirmation>
          </saml:Subject>
    </saml:AuthenticationStatement>
</saml:Assertion>
```

The elements in the assertion have the following meaning:

Tag name	Required	Description
MajorVersion	X	Major version of the assertion
MinorVersion	X	Minor version of the assertion
AssertionID	X	ID of the assertion
Issuer	X	The name of the SAML authority that created the assertion
IssuerInstant	X	Issuing time of the assertion
Conditions		Conditions that MUST be taken into account when assessing the validity of the assertion
AuthenticationMethod	X	Specifies what kind of authentication took place
AuthenticationInstant	X	Specifies the time when the authentication took place
Qualifier		The name by which the subject is recognized
Format		A URI reference representing the format of NameIdentifier
SubjectConfirmation		Specifies a subject by supplying data that allows the subject to be authenticated
ConfirmationMethod		Identifies which method to be used for authenticating the subject

Figure 10: Tag descriptions

9. Requirement Comparison

A future version of this document will compare the requirements listed in [I-D.ietf-sipping-trait-authz] with the solution provided in this document.

10. Security Considerations

This section discusses security considerations when using SAML with SIP.

10.1 Stolen Assertion

Threat:

If an eavesdropper can copy the real user's SAML response and included assertions and construct a SIP message of his own, then the eavesdropper could be able to impersonate the user at other SIP entities.

Countermeasures:

By providing adequate confidentiality, eavesdropping of a SAML assertion can be stopped.

10.2 MitM Attack

Threat:

Since the SAML assertion is carried within a SIP message, a malicious site could impersonate the user at some other SIP entities. These SIP entities would believe the adversary to be the subject of the assertion.

Countermeasures:

If the adversary is a not-participating in the SIP signaling itself (i.e., it is not a SIP proxy or a SIP UA), this threat can be eliminated by employing inherent SIP security mechanisms, such as TLS. However, if this entity is part of the communication itself then reference integrity needs to be provided. Assertions with tight restrictions (e.g., validity of the assertion) can also limit the possible damage.

10.3 Forged Assertion

Threat:

A malicious user could forge or alter a SAML assertion in order to communicate with the SIP entities.

Countermeasures:

To avoid this kind of attack, the entities must assure that proper mechanisms for protecting the SAML assertion needs to be in place. It is recommended to protect the assertion using a digital signature.

10.4 Replay Attack

Threat:

In the case of using SIP with the SAML pull model, the threat of replay lies in the fact that the artifact is a one-time-use subject. The same artifact can be used again to gain access to resources.

Countermeasures:

Cases where multiple requests are made which references the same request must be tracked in order to avoid the threat.

11. Contributors

The authors would like to thank Henning Schulzrinne for his contributions to this document.

12. Acknowledgments

We would like to thank RL 'Bob' Morgan and Stefan Goeman for their comments to this draft.

13. IANA Considerations

This document contains a number of IANA considerations. A future version of this document will list them in this section.

14. Open Issues

This document raises a number of issues with regard to the SIP protocol interaction. Some of them are raised in this document (such as reference integrity, who is allowed to add which information, etc.) but a more detailed treatment of this topic with a focus of identity management is described in [I-D.peterson-message-identity]. In particular, the following sections are highly relevant for this document:

Assertion Constraints and Scope:

This aspect deals with the problem of binding a SIP assertion to a specific SIP message. The goal is to provide a security property called reference integrity to prevent replay and impersonation attacks. As described in Section 5 that a number of fields can be used for this purpose. This document also considers scenarios where the SAML assertion was obtained outside a SIP protocol run. In these cases SIP fields are not available to provide reference integrity. The concept of the holder-of-the-key assertion is described below and relevant for this discussion.

Canonicalization versus Replication:

To provide reference integrity a few selected fields need to be hashed, signed and placed into the assertion. Two approaches are available for this purpose. Hence it needs to be studied how the interworking between reference integrity and the usage of obtained SAML assertion can be properly accomplished. For example, who indicates which fields are included?

Placement of Assertions and Keys in Messages:

This document assumes that the assertions are added to the SIP body and artifacts or references to assertions located in the SIP body are added to the SIP header. A central question is therefore where these assertions should be attached? Should the SIP user agent or intermediate SIP proxies add assertions/artifacts? In the scenarios depicted in Section 6, we have both approaches depending on what kind of scenario it is. In Figure 4, they are added at the UA and in contrast we have Figure 7, where the assertions are added at the PSTN/SIP gateway.

MIME bodies can only be attached at the UA. Proxies by definition do not attach MIME bodies; if an intermediary were to do so, it would not be playing the proxy server role in the SIP architecture. The SIP content indirection mechanism [I-D.ietf-sip-content-indirect-mech] is also relevant in this discussion.

To provide reference integrity (by binding a SIP session and a SAML assertion together) two alternative approaches exist:

Hashing of SIP message fields:

If a hash is computed over a number of selected SIP fields and subsequently digitally signed then there is a some degree of protection that the assertion cannot be copied to other SIP messages and reused. The drawback thereby is that the assertion has a very limited time period. The hashed fields may vary from context to context.

Holder-of-the-Key Assertion:

SAML introduces the concept of a holder-of-the-key assertion to bind the assertions (authorization information) to a cryptographic key. As a result, the end host which was quite passive when dealing with assertions can be turned into an active protocol participant. The end host obtained the assertion and attached them to selected messages but did not provide any cryptographic computations with regard to the assertion itself. With the end host being active in the protocol exchange security is improved a lot. Furthermore, it is possible to combine the benefits of the work done with SIPPING-CERT [I-D.ietf-sipping-certs] and this document by binding a self-signed certificate created by the user and stored at the credential server to an assertion. As noted in Section 9 of [I-D.ietf-sipping-certs] in the context of signing SIP messages the usage of a self-signed certificate is not very helpful except used with an Authentication Service. Combined with a SAML assertion the signature would protect the SIP message and the SAML assertion would provide authorization information.

A number of credentials can be used with the KeyInfo element of the Holder-of-the-Key assertion as described in Section 4.4 of [xmldsig-core].

Further open issues are:

- Some work on option-tags is required. Are there cases when processing of the assertion would be required by the sender? Or when a proxy server wants to be able to say that a UA must supply this header in order to access a particular resource? If so, an option-tag should be defined for this extension that can be used in Require, Supported, 420, etc.

- Specific SAML confirmation method identifiers and identifiers for the bindings or profiles must be defined and registered with OASIS. A confirmation method identifier is a URI that specifies

which method should be used by the target domain to assure that the identity of the subject is true.

This mechanism seems to be provide the same reference integrity properties as the hash over the various headers/bodies proposed in the identity draft.

- Further use cases would be interesting. For example, a mechanism to provide additional security for the SIP REFER method [RFC3515].

- A few new URIs need to be registered. The proposed URIs for identification are:

  SIP Binding: urn:oasis:names:tc:SAML:1.0:bindings:SIP-binding

  Artifact
  profile: urn:oasis:names:tc:SAML:1.0:profiles:SIP-artifact-01

  Assertion
  profile: urn:oasis:names:tc:SAML:1.0:profiles:SIP-assertion-01

- The proposed URIs for Confirmation Method Identifiers are:

  Artifact profile: urn:oasis:names:tc:SAML:1.0:cm:SIP-artifact-01

  Assertion profile: urn:oasis:names:tc:SAML:1.0:cm:SIP-bearer

- These are based on the URIs already used in the existing SOAP-SAML binding, specified in Section 3 of [I-D.saml-bindings-1.1].

- An alignment with the work done by Liberty Alliance on Federated Identities as described in [I-D.liberty-idff-arch-overview] would be useful.

- The security consideration needs more details.

15. References

15.1 Normative References

[I-D.hodges-saml-mediatype]
Hodges, J., "application/saml+xml Media Type Registration", draft-hodges-saml-mediatype-01 (work in progress), June 2004.

[I-D.ietf-sipping-trait-authz]
Peterson, J., "Trait-based Authorization Requirements for the Session Initiation Protocol (SIP)", draft-ietf-sipping-trait-authz-01 (work in progress), February 2005.

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

[RFC2392] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998.

15.2 Informative References

[I-D.ietf-sip-authid-body]
Peterson, J., "SIP Authenticated Identity Body (AIB) Format", draft-ietf-sip-authid-body-03 (work in progress), May 2004.

[I-D.ietf-sip-content-indirect-mech]
Burger, E., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", draft-ietf-sip-content-indirect-mech-05 (work in progress), October 2004.

[I-D.ietf-sip-identity]
Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[I-D.ietf-sipping-certs]
Jennings, C. and J. Peterson, "Certificate Management Service for The Session Initiation Protocol (SIP)", draft-ietf-sipping-certs-01 (work in progress), February 2005.

[I-D.jennings-sipping-pay]
Jennings, C., "Payment for Services in Session Initiation

Protocol (SIP)", draft-jennings-sipping-pay-01 (work in progress), February 2005.

[I-D.liberty-idff-arch-overview]
Wason, T., "Liberty ID-FF Architecture Overview", 2004.

[I-D.peterson-message-identity]
Peterson, J., "Security Considerations for Impersonation and Identity in Messaging Systems", draft-peterson-message-identity-00 (work in progress), October 2004.

[I-D.saml-bindings-1.1]
Maler, E., Philpott, R., and P. Mishra, "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1", September 2003.

[I-D.saml-core-1.1]
Maler, E., Philpott, R., and P. Mishra, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1", September 2003.

[I-D.saml-sec-consider-1.1]
Maler, E. and R. Philpott, "Security and Privacy Considerations for the OASIS Security Markup Language (SAML) V1.1", September 2003.

[I-D.saml-tech-overview-1.1-03]
Maler, E. and J. Hughes, "Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1", March 2004.

[RFC2543] Handley, M., Schulzrinne, H., Schooler, E., and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[xmldsig-core]
Eastlake, D., Reagle, J., and D. Solo, "XML-Signature Syntax and Processing, W3C Recommendation (available at http://www.w3.org/TR/xmldsig-core/)", February 2002.

Authors' Addresses

Hannes Tschofenig
Siemens
Otto-Hahn-Ring 6
Munich, Bavaria 81739
Germany

Email: Hannes.Tschofenig@siemens.com

Jon Peterson
NeuStar, Inc.
1800 Sutter St Suite 570
Concord, CA 94520
US

Email: jon.peterson@neustar.biz

James Polk
Cisco
2200 East President George Bush Turnpike
Richardson, Texas 75082
US

Email: jmpolk@cisco.com

Douglas C. Sicker
University of Colorado at Boulder
ECOT 430
Boulder, CO 80309
US

Email: douglas.sicker@colorado.edu

Marcus Tegnander
Letterkenny Institute of Technology
Port Road
Letterkenny, Donegal
Ireland

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP Working Group
Internet-Draft
Expires: December 16, 2005

D. Willis, Ed.
Cisco Systems
A. Allen
Research in Motion (RIM)
June 14, 2005

Requesting Answering and Alerting Modes for the Session Initiation Protocol (SIP)
draft-willis-sip-answeralert-00

Status of this Memo

Copyright Notice

Abstract

This document defines two SIP extension header fields and associated option tags that can be used in INVITE requests to convey the requester's preference for user-interface handling of that request. The first header, "Answer-Mode", expresses a preference as to whether the target node's user interface waits for user input before accepting the request or instead accepts the request without waiting

on user input. The second header, "Alert-Mode", expresses a preference as to whether the target node's user interface alerts the user about the request. These behaviors have applicability to applications such as Push-to-Talk and to diagnostics like loop-back. This document also defines use of the SIP extension header field "Answer-Mode", in a response to an INVITE request to inform the requester as to which answer mode was actually applied to this request. There are significant security considerations, especially when the two request options are used together.

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 [1].

Table of Contents

1. Background

There has been discussion of how to deal with "auto-answer" and related issues in the SIP community for several years. Discussion in the SIPPING working group, augmented by input from other organizations such as the Open Mobile Alliance, resulted in a consensus observed in the SIPPING meeting at IETF 62 to extend SIP, which is defined in [2]. Further discussion of the topic on the SIP mailing list after IETF 62 led to a consensus to pursue this work in the SIP working group as a standards-track effort.

Two different use cases converged to create the consensus for the development of this specification. Other use cases presumably exist, but two is enough to establish the level of reusability required to justify a standards-track extension as opposed to a "P-header" under [3].

The first key use case was the requirement for diagnostic loopback calls. In this sort of scenario, a testing service sends an INVITE to a node being tested. The tested node accepts and a dialog is established. But rather than establishing a two-way media flow, the tested node loops back or "echoes" media received from the testing service back toward the testing service. The testing service can then analyze the media flow for quality and timing characteristics. SDP usage for this sort of flow is described in [11]. In this sort of application, it may not be needful that the human using the node under test interact with the node in any way for the test to be satisfactorily executed. In some cases, it might be appropriate to alert the user to the ongoing test, and in other cases it might not be.

The second use case is that of "Push to Talk" applications as described in [12] and relates to a service being specified by the Open Mobile Alliance. In this sort of environment, SIP is used to establish ‘a dialog supporting asynchronous delivery of unidirectional media flow, giving a user experience like that of a traditional two-way radio. It is conventional for the INVITES used to be automatically accepted by the called UA (User Agent), and the media is commonly played out on a loudspeaker.

These sorts of mechanisms are not required to provide the functionality of an "answering machine" or "voice mail recorder". Such a device knows that it should answer and does not require a SIP extension to support its behavior.

Much of the discussion of this topic in working group meetings and on the mailing list dealt with disambiguating "answering mode" from "alerting mode". Some early work, such as [12], did not make this

distinction. We therefore proceed with the following definitions:

- Answering Mode includes behaviors in a SIP UA relating to acceptance or rejection of a request that are contingent on interaction between the UA and the user of that UA after the UA has received the request. We are principally concerned with the user interaction involved in accepting the request and initiating an active session. An example of this might be pressing the "yes" button on a mobile phone.
- Alerting Mode includes behaviors in a SIP UA relating to to informing the user of the UA that a request to initiate a session has been received. An example of this might be activating the ring tone of a mobile phone.

2. Requirements

Requirements in the following are expressed relative to the node initiating an INVITE request (UAC), the node receiving and potentially responding to that request (UAS), and the users of those nodes (UAC-user and UAS-user).

2.1 Requirements for Requesting an Answering Mode

The requirements relating to requesting a specific answering mode include:

Req-1: It MUST be possible for UAC to ask that the UAS answer the request without requiring interaction between UAS-user and the user interface (UI) of the UAS. We refer to this as "automatic answer mode". This mode is useful for diagnostic loopback procedures and critical for "two-way radio" or "push to talk" applications.

Req-2: It MUST be possible for UAC to ask that the UAS answer the request only after UAS-user has directed UAS to answer this specific request. We refer to this as "manual answer mode". This mode is useful in "push to talk" applications where the sender requires a reassurance that somebody is listening.

Req-3: It MUST be possible for UAS to apply local policy to each request and determine whether or not to provide the requested answer mode for this request. This policy determination MAY include authentication checks, authorization against "buddy lists" as used in some presence systems, or other mechanisms outside the scope of this specification. This behavior is critical in avoiding major security pitfalls, such as turning the victim's phone into a "bug" or eavesdropping device.

Req-4: It MUST be possible for UAC to indicate in the request that this extension for selecting answering mode is required, such that UAS MUST reject the request if it does not support this extension. This can be used to prevent automated diagnostic loopback requests from annoying nodes not supporting this extension

Req-5: It MUST be possible for UAC to indicate at least two different priority levels for the desired answer mode. We refer to these as "normal" and "override" priorities. In normal usage, we expect that "normal" priority would be used in a user-to-user fashion, whereas "override" priorities would be used for diagnostic procedures or some sorts of emergency session establishment. This behavior allows a device to be set up such that it might not auto-answer routine calls, but could be convinced to auto-answer an emergency or other high-priority call.

Req-6: It MUST be possible for UAS or proxies acting on behalf of UAS to apply policy relative to the indicated priority level. This MAY include having different authentication and or authorization procedures for each priority level. This capability allows functions like time-of-day call screening, so that routine calls that would normally be rejected locally by the device would be blocked by a proxy without access network costs, but high-priority calls that would override routine call screening could be passed to the device.

Req-7: It MUST be possible for UAS to indicate its support for the selection of answer modes in a REGISTER request so that that the routing proxy can selectively route requests requiring the selection of answer mode to UAS. This requirement enables the functions described in the next requirement.

Req-8: It MUST be possible for the UAC to construct the request in such a way that the routing proxy infrastructure, if present, will select only contacts supporting the selection of answer modes. This can efficiently (minimal access network traffic and minimal forking load) prevent devices that do not support this extension from being reached by requests that require this extension. Note that this requirement does NOT include selection of a singular UAS from a set to which the request might be forked.

Req-9: It MUST be possible for UAC to discover whether UAS supports the selection of answer modes via a SIP OPTIONS request.

Req-10: It MUST be possible for an intermediate proxy acting on behalf of UAC or UAS to apply policy relative to the answer mode indicated in a request. For example, a proxy may require special authentication and authorization for a request that places a high priority on auto-answer capabilities. Application of policy here means altering the requested answer mode and/or inserting or deleting a request for a specific answer mode.

2.2 Requirements for Requesting an Alerting Mode

The requirements relating to requesting a specific alerting mode include:

Req-11: It MUST be possible for UAC to ask that UAS answer the request without alerting UAS-user. This allows for diagnostic loopbacks that do not needlessly interrupt the user of a device.

Req-12: It MUST be possible for UAS to apply local policy to each request and determine whether or not to provide the requested alerting mode for this request. This policy determination MAY include authentication checks, authorization against "buddy lists" as used in some presence systems, or other mechanisms outside the scope of this specification.

Req-13: It MUST be possible for UAC to indicate in the request that this extension for selecting alerting mode is required, such that UAS MUST reject the request if it does not support this extension. This capability augments the ability of automated testing functions to operate non-intrusively when some devices in a network do not support this extension.

Req-14: It MUST be possible for UAC to discover whether UAS supports the selection of alerting modes via a SIP OPTIONS request.

Req-15: It MUST be possible for UAS to indicate its support for the selection of alerting modes in a REGISTER request so that that the routing proxy can selectively route requests requiring the selection of alerting mode to UAS. This supports the functionality described in the following requirement.

Req-16: It MUST be possible for UAC to construct the request in such a way that the routing proxy infrastructure, if present, will select only contacts supporting the selection of alerting modes. This allows the proxy network to efficiently avoid sending the request to nodes that do not support this extension.

Req-17: It MUST be possible for an intermediate proxy acting on behalf of UAC or UAS to apply policy relative to the alerting mode indicated in a request. Application of policy here means altering the requested alerting mode and/or inserting or deleting a request for a specific alerting mode.

2.3 Requirements for Indicating the Applied Answer Mode in a Response

The requirements relating to indicating which answering mode applied to the request include:

Req-18: It MUST be possible for UAS when sending a positive response to a request to indicate the answering mode that applied to the request. This allows UAC to inform UAC-user as to whether the request was answered automatically or as a result of user interaction, knowledge that may be important in informing UAC-

user's usage of the session.
Req-19: UAS SHOULD accurately represent the answering mode that was applied, but MAY either not include this information or MAY misinform UAC in order to maintain the privacy expectations of UAS-user. Consequently, applications MUST NOT rely on the veracity of this information.

3. Syntax of Header Fields and Tags

3.1 Syntax of Header Field and Tags

The syntax for the header fields defined in this document is:

```
Answer-Mode = "Answer-Mode" HCOLON answer-mode
answer-mode = "Manual" / "ManualReq" / "Auto" / "AutoReq"
Alert-Mode = "Alert-Mode" HCOLON alert-mode
alert-mode = "Normal" / "Null"
```

The syntax of the Alert-Mode option tag is:

```
Alert-Mode = "alertmode"
```

The syntax of the Answer-Mode option tag is:

```
Answer-Mode = "answermode"
```

The syntax of the feature tag indicating support for selection of the answer mode is:

```
Answer-Mode = "answermode"
```

The syntax for feature tags is defined in [4]

The value range of the Answer-Mode feature tag is binary, with values of "TRUE" or "FALSE".

3.2 Amendments to Table 2 and 3 of RFC3261

The allowable usage of header fields is described in Tables 2 and 3 of [2]. The following additions to this table are needed for the extension header fields defined in this document.

Additions to SIP Table 3:

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	PRA
Answer-Mode	I	adm	-	-	-	-	-	-	-
Alert-Mode	I	adm	-	-	-	-	-	-	-
Answer-Mode	200		-	-	-	X	-	-	-

Figure 1

4. Usage of the Answer-Mode Header Field, Option, and Media Feature Tags in a Request

The Answer-Mode header field is used by a UAC to request specific handling of an INVITE request by the responding UAS related to "automatic answering" functionality. If no Answer-Mode header field is included in the request, answering behavior is at the discretion of the UAS, as it would be in the absence of this specification. The desired handling is indicated by the the value of the Answer-Mode header field, as follows:

Manual: The UAS is asked to not accept the request (send a 200 OK) until the user of the UAS has interacted with the user interface (UI) of the UAS in such a way as to indicate that the user desires the UAS to accept the request.

ManualReq: The UAS is strongly asked to accept the request manually, as in "Manual". Further, the UAS is asked to override local user preferences relating to automatic answer, and answer manually even if the user preferences are to automatically answer requests having a Answer-Mode header field value of "Manual". The UAS is also asked NOT to answer automatically, and to reject the request if it is unwilling to answer manually.

Auto: The UAS is asked to accept the request automatically, without waiting for the user of the UAS to interact with the UI of the UAS in such a way as to indicate that the user desires the UAS to accept the request.

AutoReq: The UAS is strongly asked to accept the request automatically, as in "Auto". Further, the UAS is asked to override local user preferences relating to automatic answer, and answer automatically even if the user preferences are to not automatically answer requests having a Answer-Mode header field value of "Auto". The UAS is also asked NOT to answer manually, and to reject the request if it is unwilling to answer automatically.

4.1 Procedures at the UAC

4.1.1 All Requests

A UAC supporting this specification indicates its support for this extension by including an option tag of "answermode" in the Supported header field of all requests it sends.

4.1.2 REGISTER Transactions

To indicate that it supports the answer-mode negotiation feature, a

UA includes a SIP extension feature tag of "answermode" in the Contact: header field of its REGISTER requests. This usage of feature tags is described in [5].

4.1.3 INVITE Transactions

A UAC supporting this specification includes a Answer-Mode header field and appropriate value in an INVITE where it wishes to influence the answering mode of the responding UAS.

To request that the UAS answer only after having interacted with its user and receiving an affirmative instruction from that user, the UAC includes a Answer-Mode header field having a value of "Manual".

To request that the UAS answer manually, and ask that it reject the INVITE request if unable or unwilling to answer manually, the UAC includes a Answer-Mode header field having a value of "ManualReq".

To request that the UAS answer automatically without waiting for input from the user, the UAC includes a Answer-Mode header field having a value of "Auto".

To request that the UAS answer automatically, and ask that it reject the INVITE request if unable or unwilling to answer automatically, the UAC includes a Answer-Mode header field having a value of "AutoReq".

To require that the UAS either support this extension or reject the request, the UAC includes a Required: header field having the value "answermode". Note that this does not actually force the UAS to automatically answer, it just requires that the UAS understand this negotiation mechanism. We do not have a negotiation technique (like "requires") to force specific behavior. Rather, the desired behavior is indicated in the SIP extension itself.

To request that retargeting proxies in the path preferentially select targets that have indicated support for this extension in their registration, a UAC includes an Accept-Contact header field having a parameter of "answermode". This usage of Accept-Contact is described in [6].

To request that retargeting proxies in the path do not select targets that have indicated non-support for this extension in their registration, a UAC includes an Accept-Contact header field having a parameter of "answermode" and an option field of "require". This usage of Accept-Contact is described in [6].

To request that retargeting proxies in the path exclusively select

targets that have indicated support for this extension in their registration, a UAC includes an Accept-Contact header field having a parameter of "answermode" and option fields of "require" and "explicit". This usage of Accept-Contact is described in [6].

4.2 Procedures at Intermediate Proxies

The general procedure at all intermediate proxies including the UAC's serving proxy or proxies and the UAS's serving proxy or proxies is to ignore the Answer-Mode header field. However, the serving proxies MAY exercise control over the requested answer mode, either inserting or deleting a Answer-Mode header field or altering the value of an existing header field in accord with local policy. Note that this may result in behavior that is inconsistent with user expectations, such as having a call that was intended to be a diagnostic loopback answered by a human, and consequently must be done very carefully. These serving proxies MAY also reject a request according to local policy, and SHOULD use the rejection codes as specified below for the UAS if they do so.

4.3 Procedures at the UAS

For a request having an Answer-Mode value of "Manual", the UAS SHOULD defer accepting the request until the user of the UAS has confirmed willingness to accept the request. This behavior MAY be altered as needed for unattended UAS or other local characteristics or policy. For example, an auto-attendant system that always answers automatically would go ahead and answer, despite the presence of the "Manual" Answer-Mode header field value.

For a request having an Answer-Mode value of "ManualReq", the UAS SHOULD defer accepting the request until the user of the UAS has confirmed willingness to accept the request. If the UAS is not capable of answering the request in this "Manual" mode or is unwilling to do so, it SHOULD reject the request with a "403 Forbidden" response and MAY include a Reason [7] header field value of:

Reason: SIP ;cause=403 ;text="manual answer forbidden"

For a request having an Answer-Mode value of "Auto", the UAS SHOULD, if the calling party is authenticated and authorized for automatic answering, accept the request without further user input. The UAS MAY, according to local policy or user preferences, treat this request as it would treat a request having a Answer-Mode with a value of "Manual" or having no Answer-Mode header field. If the calling party is not authenticated and authorized for automatic answer, the UAS may either handle the request as per "manual", or reject the

request. If the UAS rejects the request, it SHOULD do so with a "403 Forbidden" response, and MAY include a Reason [7] header field value of:

Reason: SIP ;cause=403 ;text="automatic answer forbidden"

For a request having an Answer-Mode value of "AutoReq", the UAS SHOULD apply authentication and authorization checks before accepting such a request. The UAS MUST NOT allow "manual" answer of this request, but MAY reject it. If, for whatever reason, the UAS chooses not to accept the request automatically, the UAS MUST reject the request and SHOULD do so with a "403 Forbidden" response, and MAY include a Reason [7] header field value of:

Reason: SIP ;cause=403 ;text="automatic answer forbidden"

4.4 Issues with Automatic Answering and Forking

One of the well-known issues with forking is the problem of multiple acceptance. If an INVITE request is forked to several UAS, and more than one of those UAS respond with a 200 OK, the conventional approach is to continue the dialog with the first respondent, and tear down the dialog (via BYE) with all other respondents.

While this problem exists without an auto-answer negotiation capability, it is apparent that widespread adoption of UAS that engage in auto-answer behavior will exacerbate the multiple acceptance problem. Consequently, systems designers need to take this aspect into consideration. In general, auto-answer is probably NOT RECOMMENDED in environments that include forking.

As an alternative, it might be reasonable to use a variation on manual-answer combined with no alerting and early media. In this approach, the initial message or talk-burst is transmitted as early media to all recipients, where it is displayed or played out. Any response utterance from the user of a UAS following this would serve as an "acceptance", resulting in a 200 OK response being transmitted by their UAS. Consequently, the race-condition for acceptance would be limited to the subset of UAs actually responding under user control, rather than the full set of UAS to which the request was forked.

Another alternative would be to use dynamic conferencing instead of forking. In this approach, instead of forking the request, a conference would be initiated and all UAs invited into that conference. The mixer attached to the conference would then mediate traffic flows appropriately.

5. Usage of the Alert-Mode Header Field, Option, and Media Feature Tags In a Request

The Alert-Mode header field is used by a UAC to request specific handling of an INVITE request by the responding UAS related to the alerting of the user of the UAS. If no Alert-Mode header field is included in the request, alerting behavior is at the discretion of the UAS, as it would be in the absence of this specification. The desired handling is indicated by the the value of the Alert-Mode header field, as follows:

Normal: The UAS is asked to treat the request as it normally would in the absence of this specification and exercise whatever alerting mechanism it might have and be configured to use.
Null: The UAS is asked to not alert its user to the request.

5.1 Procedures at the UAC

5.1.1 All Requests

A UAC supporting this specification indicates its support for this extension by including an option tag of "answermode" in the Supported header field of all requests it sends.

5.1.2 REGISTER Transactions

To indicate that it supports the alert-mode negotiation feature, a UA includes a SIP extension feature tag of "alertmode" in the Contact: header field of its REGISTER requests. This usage of feature tags is described in [5].

5.1.3 INVITE transactions

A UAC supporting this specification includes a Alert-Mode header field and appropriate value in an INVITE where it wishes to influence the alerting mode of the responding UAS.

To request that the UAS not alert its user the UAC includes a Alert-Mode header field having a value of "Null".

To request that the UAS apply its normal procedures for alerting the user the UAC either includes a Alert-Mode header field having a value of "Normal" or it includes no Alert-Mode header field.

To require that the UAS either support this extension or reject the request, the UAC includes a Required: header field having a value of "alertmode".

5.2 Procedures at Intermediate Proxies

The general procedure at all intermediate proxies including the UAC's serving proxy or proxies and the UAS's serving proxy or proxies is to ignore the Alert-Mode header field. However, the serving proxies MAY exercise control over the requested answer mode, either inserting or deleting a Alert-Mode header field or altering the value of an existing header field in accord with local policy. Note that this may result in behavior that is inconsistent with user expectations, such as having a call that was intended to be a silent diagnostic loopback answered by a human, and consequently must be done very carefully. These serving proxies MAY also reject a request according to local policy, and SHOULD use the rejection codes as specified below for the UAS if they do so.

5.3 Procedures at the UAS

A UAS supporting this specification considers the value of the Alert-Mode header field in an INVITE request in determining how and/or whether to alert the user of the UAS to the request. The UAS may also consider local policy, the presence of an authenticated identity or other authentication, and other elements of the request in making this determination.

If the conclusion is to alert the user, the UAS invokes its preferred alerting mechanism. If the conclusion is to not alert the user, the UAS proceeds to process the request. Note that the decision of whether to accept the request is independent of the alerting decision, but one can generally not expect the user to make this decision unless the user has been alerted to the request.

The general intent of a request having a Alert-Mode header field with a value of "Null" is that the user not be invasively interrupted by the request. Consequently, it might be appropriate to invoke a less-disruptive alerting mechanism (perhaps blinking a small light) as an alternative to not invoking any alerting mechanism.

6. Usage of the Answer-Mode Header Field in a Response

The Answer-Mode header field may be inserted by a UAS into a response in order to indicate how it handled the associated request with respect to automatic answering functionality. The UAC may use this information to inform the user or otherwise adapt the behavior of the user interface. The handling is indicated by the the value of the Answer-Mode header field, as follows:

Manual: The UAS responded after the user of the UAS interacted with the user interface (UI) of the UAS in such a way as to indicate that the user desires the UAS to accept the request.
ManualReq: The UAS responded manually, as above. Further, the request contained a Answer-Mode header field with the value "ManualReq", and the UAS has honored this requirement.
Auto: The UAS responded automatically, without waiting for the user of the UAS to interact with the UI of the UAS in such a way as to indicate that the user desires the UAS to accept the request.
AutoReq: The UAS responded automatically (as above). Further, the request contained a Answer-Mode header field with the value "AutoReq", and the UAS has honored this requirement.

The Answer-Mode header field, when used in a response, is only valid in a 200 OK response to an INVITE request.

6.1 Procedures at the UAS

A UAS supporting this specification inserts a Answer-Mode header field into the 200 OK response to an INVITE request when it wishes to inform the UAC as to whether the request was answered manually or automatically. The full rationale for including or not including this header field in a response is outside of the scope of this specification. However, it is reasonable for a UAS to assume that if the UAC included a Answer-Mode header field in the request that it would probably like to see a Answer-Mode header field in the response.

6.2 Procedures at the UAC

A UAC can use the value of the Answer-Mode header field, if present, to adapt the user interface and/or inform the user about the handling of the request. For example, the user of a push-to-talk system might speak differently if she knows that the called party answered "in person" vs. having the call blare out of an unattended speaker phone.

7. Examples of Usage

The following examples show Bob registering a contact that supports negotiation of answer mode and alerting mode. Alice then calls Bob with an an INVITE request, asking for automatic answering with normal alerting and explicitly asking that the request not be routed to contacts that have not indicated support for this extension. Further, Alice requires that the request be rejected if Bob's UA does not support negotiation of alerting and answer modes. Bob responds with a 200 OK indicating that the call was answered automatically.

7.1 REGISTER Request

```
REGISTER sip:example.com SIP/2.0
From: Bob <sip:bob@example.com>
To: Bob <sip:bob@example.com>
Contact: sip:cell-phone@example.com;
+sip.extensions="answermode";
methods="INVITE,BYE,OPTIONS,CANCEL,ACK"
```

7.2 INVITE Request

```
INVITE <sip:bob@example.com SIP/2. 0>
Via: SIP/2.0/TCP client-alice.example.com:5060;branch=z9hG4bK74b43
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@example.com>
Call-ID:3848276298220188511@client-alice.example.com
CSeq: 1 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Requires: answermode, alertmode
Accept-contact:*;require;explicit;
+sip.extensions="answermode";
+sip.extensions="alertmode";
Answer-Mode: Auto
Alert-Mode: Null
Content-Type: application/sdp
Content-Length: ...
```

7.3 200 OK response

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP client-alice.example.com:5060;branch=z9hG4bK74bf9
From: Alice <sip:alice@example.com>;tag=9fxced76sl
To: Bob <sip:bob@example.com>;tag=8321234356
Call-ID: 3848276298220188511@client-alice.example.com
CSeq: 1 INVITE
Contact: <sip:bob@client.biloxi.example.com;transport=tcp>
Answer-Mode: Auto
Content-Type: application/sdp
Content-Length: ...
```

8. Security Considerations

This specification adds the ability for a UAC to request potentially risky user interface behavior relating to the acceptance of an INVITE request by the UAS receiving the request. These behaviors include accepting the request without notification of the user of the UAS, and accepting the request without input to the UAS by the user of the

UAS.

There are several attacks possible here, with the most obvious being the ability to turn a phone into a remote listening device without its user being aware of it. Additional attacks include reverse charge fraud, unsolicited "push to talk" communications (SPPTT), battery-rundown denial-of-service, "forced busy" denial of service, and phishing via session insertion (where an ongoing session is replaced by another without the victim's awareness.

In the most common use cases, the security aspects are somewhat mitigated by design aspects of the application. For example, in push-to-talk applications, no media is sent from the called UA without user input (the "push" of "push-to-talk"). Consequently, there is no "bugging" attack when the "Null" Alert-Mode option is exercised in conjunction with automatic answering. Furthermore, the incoming initial talk burst, if present, may serve to alert the called user. However, there is still the potential for an "unsolicited message transmission". For example, the initial talk-burst of an auto-answered push-to-talk session might include an advertisement for pharmaceuticals, or broadcast rude noises in the tradition of the "whoopee cushion."

Consequently, the UAS generally or its supporting proxy MUST authenticate the sender of such requests, using mechanisms such as SIP Digest Authentication, [2], the SIP Identity mechanism [13], or the SIP mechanism for Asserted Identity Within Private Networks[8], in networks for which it is suitable.

The authenticated identity of the requester MUST then be matched against authorization policy appropriate to the requested application. For example, it might be appropriate to allow a designated systems administrator to start a diagnostic loopback session without alerting the user. It might also be appropriate to allow a known "buddy" to start a push-to-talk session without requiring the user of the UAS to actively accept the call. It is almost certainly NOT appropriate to allow an unauthenticated and unauthorized requester to start a session without alerting and receiving a confirmation of acceptance (manual answer) from the targeted user.

9. IANA Considerations

9.1 Registration of Header Fields

This document defines new SIP header fields named "Answer-Mode", "Alert-Mode", and "Answer-Mode".

The following rows shall be added to the "Header Fields" section of the SIP parameter registry:

Header Name	Compact Form	Reference
Answer-Mode		[RFCXXXX]
Alert-Mode		[RFCXXXX]
Answer-Mode		[RFCXXXX]

Editor Note: [RFCXXXX] should be replaced with the designation of this document.

9.2 Registration of Header Field Parameters

This document defines parameters for the header fields defined in the preceding section. The header field named "Answer-Mode" may take the values "Manual", "Auto", or "AutoReq". The header field named "Alert-Mode" may take the values "Normal" or "Null".

The following rows shall be added to the "Header Field Parameters and Parameter Values" section of the SIP parameter registry:

Header Field	Parameter Name	Predefined Values	Reference
Answer-Mode	Manual	Yes	[RFCXXXX]
Answer-Mode	Auto	Yes	[RFCXXXX]
Answer-Mode	AutoReq	Yes	[RFCXXXX]
Alert-Mode	Normal	Yes	[RFCXXXX]
Alert-Mode	Null	Yes	[RFCXXXX]

Editor Note: [RFCXXXX] should be replaced with the designation of this document.

9.3 Registration of Extension Option Tags

This document defines new SIP option tags "answermode" and "alertmode".

The following rows shall be added to the "Option Tags" section of the SIP Parameters registry:

Name	Description	Reference
answermode	This option tag is used in a Requires header field to indicate that the UAS must support negotiation of answer mode.	[RFCXXXX]
alertmode	This option tag is used in a Requires header field to indicate that the UAS must support negotiation of alerting mode.	[RFCXXXX]

Editor Note: [RFCXXXX] should be replaced with the designation of this document.

10. Acknowledgements

This document draws requirements and a large part of its methodology from the work of the Open Mobile Alliance, and specifically from the internet draft [12] by Andrew Allen, Jan Holm, and Tom Hallin.

The editor would also like to recognize the contributions of David Oran and others who argued on the SIPPING mailing list and at the OMA ad-hoc meeting at IETF 62 that the underlying ideas of the above draft were broadly applicable to the SIP community, and that the concepts of alerting and answering should be clearly delineated.

11. References

11.1 Normative References

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

[2] 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.

[3] Mankin, A., Bradner, S., Mahy, R., Willis, D., Ott, J., and B. Rosen, "Change Process for the Session Initiation Protocol (SIP)", BCP 67, RFC 3427, December 2002.

[4] Klyne, G., "A Syntax for Describing Media Feature Sets", RFC 2533, March 1999.

[5] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[6] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[7] Schulzrinne, H., Oran, D., and G. Camarillo, "The Reason Header Field for the Session Initiation Protocol (SIP)", RFC 3326, December 2002.

[8] Jennings, C., Peterson, J., and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[9] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[10] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, March 1999.

11.2 Informative References

[11] Hedayat, K., "An Extension to the Session Initiation Protocol (SIP) for Media Loopback", draft-hedayat-media-loopback-01 (work in progress), October 2004.

[12] Allen, A., "Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the Open Mobile Alliance (OMA) Push to talk over Cellular (PoC)", draft-allen-sipping-poc-p-headers-01 (work in progress), February 2005.

[13] Peterson, J., "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-04 (work in progress), February 2005.

Authors' Addresses

Dean Willis (editor)
Cisco Systems
3100 Independence Pkwy #311-164
Plano, Texas 75075
USA

Phone: unlisted
Fax: unlisted
Email: dean.willis@softarmor.com

Andrew Allen
Research in Motion (RIM)
122 West John Carpenter Parkway, Suite 430
Irving, Texas 75039
USA

Phone: unlisted
Fax: unlisted
Email: aallen@rim.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.