```
ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-
```

### 23.4.3 Tunneling Encryption

It may also be desirable to use this mechanism to encrypt a "message/sip" MIME body within a CMS EnvelopedData message S/MIME body, but in practice, most header fields are of at least some use to the network; the general use of encryption with S/MIME is to secure message bodies like SDP rather than message headers. Some informational header fields, such as the Subject or Organization could perhaps warrant end-to-end security. Headers defined by future SIP applications might also require obfuscation.

Another possible application of encrypting header fields is selective anonymity. A request could be constructed with a From header field that contains no personal information (for example, sip:anonymous@anonymizer.invalid). However, a second From header field containing the genuine address-of-record of the originator could be encrypted within a "message/sip" MIME body where it will only be visible to the endpoints of a dialog.

> Note that if this mechanism is used for anonymity, the From header field will no longer be usable by the recipient of a message as an index to their certificate keychain for retrieving the proper S/MIME key to associated with the sender. The message must first be decrypted, and the "inner" From header field MUST be used as an index.

In order to provide end-to-end integrity, encrypted "message/sip" MIME bodies SHOULD be signed by the sender. This creates a "multipart/signed" MIME body that contains an encrypted body and a signature, both of type "application/pkcs7-mime".

In the following example, of an encrypted and signed message, the text boxed in asterisks ("*") is encrypted:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Anonymous <sip:anonymous@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:pc33.atlanta.com>
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
     name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
   handling=required
Content-Length: 231

**********************************************************
* Content-Type: message/sip                              *
*                                                        *
* INVITE sip:bob@biloxi.com SIP/2.0                      *
* Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8 *
* To: Bob <bob@biloxi.com>                               *
* From: Alice <alice@atlanta.com>;tag=1928301774         *
* Call-ID: a84b4c76e66710                                *
* CSeq: 314159 INVITE                                    *
* Max-Forwards: 70                                       *
* Date: Thu, 21 Feb 2002 13:02:03 GMT                    *
* Contact: <sip:alice@pc33.atlanta.com>                  *
*                                                        *
* Content-Type: application/sdp                          *
*                                                        *
* v=0                                                    *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com    *
* s=Session SDP                                          *
* t=0 0                                                  *
* c=IN IP4 pc33.atlanta.com                              *
* m=audio 3456 RTP/AVP 0 1 3 99                          *
* a=rtpmap:0 PCMU/8000                                   *
**********************************************************
```

```
--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
   handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-
```

## 24 Examples

In the following examples, we often omit the message body and the corresponding Content-Length and Content-Type header fields for brevity.

### 24.1 Registration

Bob registers on start-up. The message flow is shown in Figure 9. Note that the authentication usually required for registration is not shown for simplicity.

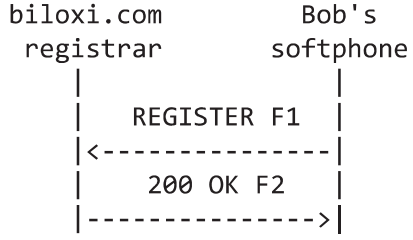


Figure 9: SIP Registration Example

F1 REGISTER Bob -> Registrar

```
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

The registration expires after two hours. The registrar responds with a 200 OK:

F2 200 OK Registrar -> Bob

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

### 24.2 Session Setup

This example contains the full details of the example session setup in Section 4. The message flow is shown in Figure 1. Note that these flows show the minimum required set of header fields - some other header fields such as Allow and Supported would normally be present.

F1 INVITE Alice -> atlanta.com proxy

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)
```

F2 100 Trying atlanta.com proxy -> Alice

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0
```

F3 INVITE atlanta.com proxy -> biloxi.com proxy

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Max-Forwards: 69
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)
```

F4 100 Trying biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0
```

F5 INVITE biloxi.com proxy -> Bob

```
INVITE sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Max-Forwards: 68
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)
```

F6 180 Ringing Bob -> biloxi.com proxy

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0
```

F7 180 Ringing biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0
```

F8 180 Ringing atlanta.com proxy -> Alice

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0
```

F9 200 OK Bob -> biloxi.com proxy

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)
```

F10 200 OK biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)
```

```
F11 200 OK atlanta.com proxy -> Alice

SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)

F12 ACK Alice -> Bob

ACK sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 ACK
Content-Length: 0

   The media session between Alice and Bob is now established.

   Bob hangs up first.  Note that Bob's SIP phone maintains its own CSeq
   numbering space, which, in this example, begins with 231.  Since Bob
   is making the request, the To and From URIs and tags have been
   swapped.

F13 BYE Bob -> Alice

BYE sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
Max-Forwards: 70
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

F14 200 OK Alice -> Bob

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

The SIP Call Flows document [40] contains further examples of SIP messages.

## 25 Augmented BNF for the SIP Protocol

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) defined in RFC 2234 [10]. Section 6.1 of RFC 2234 defines a set of core rules that are used by this specification, and not repeated here. Implementers need to be familiar with the notation and content of RFC 2234 in order to understand this specification. Certain basic rules are in uppercase, such as SP, LWS, HTAB, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions to clarify the use of rule names.

The use of square brackets is redundant syntactically. It is used as a semantic hint that the specific parameter is optional to use.

### 25.1 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986.

```
alphanum  =  ALPHA / DIGIT
```

Several rules are incorporated from RFC 2396 [5] but are updated to make them compliant with RFC 2234 [10]. These include:

```
reserved    =  ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
               / "$" / ","
unreserved  =  alphanum / mark
mark        =  "-" / "_" / "." / "!" / "~" / "*" / "'"
               / "(" / ")"
escaped     =  "%" HEXDIG HEXDIG
```

SIP header field values can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream. This is intended to behave exactly as HTTP/1.1 as described in RFC 2616 [8]. The SWS construct is used when linear white space is optional, generally between tokens and separators.

```
LWS  =  [*WSP CRLF] 1*WSP ; linear whitespace
SWS  =  [LWS] ; sep whitespace
```

To separate the header name from the rest of value, a colon is used, which, by the above rule, allows whitespace before, but no line break, and whitespace after, including a linebreak. The HCOLON defines this construct.

```
HCOLON  =  *( SP / HTAB ) ":" SWS
```

The TEXT-UTF8 rule is only used for descriptive field contents and values that are not intended to be interpreted by the message parser. Words of *TEXT-UTF8 contain characters from the UTF-8 charset (RFC 2279 [7]). The TEXT-UTF8-TRIM rule is used for descriptive field contents that are n t quoted strings, where leading and trailing LWS is not meaningful. In this regard, SIP differs from HTTP, which uses the ISO 8859-1 character set.

```
TEXT-UTF8-TRIM  =  1*TEXT-UTF8char *(*LWS TEXT-UTF8char)
TEXT-UTF8char   =  %x21-7E / UTF8-NONASCII
UTF8-NONASCII   =  %xC0-DF 1UTF8-CONT
                /  %xE0-EF 2UTF8-CONT
                /  %xF0-F7 3UTF8-CONT
                /  %xF8-Fb 4UTF8-CONT
                /  %xFC-FD 5UTF8-CONT
UTF8-CONT       =  %x80-BF
```

A CRLF is allowed in the definition of TEXT-UTF8-TRIM only as part of a header field continuation. It is expected that the folding LWS will be replaced with a single SP before interpretation of the TEXT-UTF8-TRIM value.

Hexadecimal numeric characters are used in several protocol elements. Some elements (authentication) force hex alphas to be lower case.

```
LHEX  =  DIGIT / %x61-66 ;lowercase a-f
```

Many SIP header field values consist of words separated by LWS or special characters. Unless otherwise stated, tokens are case-insensitive. These special characters MUST be in a quoted string to be used within a parameter value. The word construct is used in Call-ID to allow most separators to be used.

```
token       =  1*(alphanum / "-" / "." / "!" / "%" / "*"
               / "_" / "+" / "`" / "'" / "~" )
separators  =  "(" / ")" / "<" / ">" / "@" /
               "," / ";" / ":" / "\" / DQUOTE /
               "/" / "[" / "]" / "?" / "=" /
               "{" / "}" / SP / HTAB
word        =  1*(alphanum / "-" / "." / "!" / "%" / "*" /
               "_" / "+" / "`" / "'" / "~" /
               "(" / ")" / "<" / ">" /
               ":" / "\" / DQUOTE /
               "/" / "[" / "]" / "?" /
               "{" / "}" )
```

When tokens are used or separators are used between elements, whitespace is often allowed before or after these characters:

```
STAR    =  SWS "*" SWS ; asterisk
SLASH   =  SWS "/" SWS ; slash
EQUAL   =  SWS "=" SWS ; equal
LPAREN  =  SWS "(" SWS ; left parenthesis
RPAREN  =  SWS ")" SWS ; right parenthesis
RAQUOT  =  ">" SWS ; right angle quote
LAQUOT  =  SWS "<"; left angle quote
COMMA   =  SWS "," SWS ; comma
SEMI    =  SWS ";" SWS ; semicolon
COLON   =  SWS ":" SWS ; colon
LDQUOT  =  SWS DQUOTE; open double quotation mark
RDQUOT  =  DQUOTE SWS ; close double quotation mark
```

Comments can be included in some SIP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition. In all other fields, parentheses are considered part of the field value.

```
comment  =  LPAREN *(ctext / quoted-pair / comment) RPAREN
ctext    =  %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII
            / LWS
```

ctext includes all chars except left and right parens and backslash. A string of text is parsed as a single word if it is quoted using double-quote marks. In quoted strings, quotation marks (") and backslashes (\) need to be escaped.

```
quoted-string  =  SWS DQUOTE *(qdtext / quoted-pair ) DQUOTE
qdtext         =  LWS / %x21 / %x23-5B / %x5D-7E
                  / UTF8-NONASCII
```

The backslash character ("\") MAY be used as a single-character quoting mechanism only within quoted-string and comment constructs. Unlike HTTP/1.1, the characters CR and LF cannot be escaped by this mechanism to avoid conflict with line folding and header separation.

```
quoted-pair  =  "\" (%x00-09 / %x0B-0C
                / %x0E-7F)

SIP-URI          =  "sip:" [ userinfo ] hostport
                    uri-parameters [ headers ]
SIPS-URI         =  "sips:" [ userinfo ] hostport
                    uri-parameters [ headers ]
userinfo         =  ( user / telephone-subscriber ) [ ":" password ] "@"
user             =  1*( unreserved / escaped / user-unreserved )
user-unreserved  =  "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password         =  *( unreserved / escaped /
                    "&" / "=" / "+" / "$" / "," )
hostport         =  host [ ":" port ]
host             =  hostname / IPv4address / IPv6reference
hostname         =  *( domainlabel "." ) toplabel [ "." ]
domainlabel      =  alphanum
                    / alphanum *( alphanum / "-" ) alphanum
toplabel         =  ALPHA / ALPHA *( alphanum / "-" ) alphanum
```

```
IPv4address    =  1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference  =  "[" IPv6address "]"
IPv6address    =  hexpart [ ":" IPv4address ]
hexpart        =  hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq         =  hex4 *( ":" hex4)
hex4           =  1*4HEXDIG
port           =  1*DIGIT
```

The BNF for telephone-subscriber can be found in RFC 2806 [9]. Note, however, that any characters allowed there that are not allowed in the user part of the SIP URI MUST be escaped.

```
uri-parameters    =  *( ";" uri-parameter)
uri-parameter     =  transport-param / user-param / method-param
                     / ttl-param / maddr-param / lr-param / other-param
transport-param   =  "transport="
                     ( "udp" / "tcp" / "sctp" / "tls"
                     / other-transport)
other-transport   =  token
user-param        =  "user=" ( "phone" / "ip" / other-user)
other-user        =  token
method-param      =  "method=" Method
ttl-param         =  "ttl=" ttl
maddr-param       =  "maddr=" host
lr-param          =  "lr"
other-param       =  pname [ "=" pvalue ]
pname             =  1*paramchar
pvalue            =  1*paramchar
paramchar         =  param-unreserved / unreserved / escaped
param-unreserved  =  "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers         =  "?" header *( "&" header )
header          =  hname "=" hvalue
hname           =  1*( hnv-unreserved / unreserved / escaped )
hvalue          =  *( hnv-unreserved / unreserved / escaped )
hnv-unreserved  =  "[" / "]" / "/" / "?" / ":" / "+" / "$"

SIP-message    =  Request / Response
Request        =  Request-Line
                  *( message-header )
                  CRLF
                  [ message-body ]
Request-Line   =  Method SP Request-URI SP SIP-Version CRLF
Request-URI    =  SIP-URI / SIPS-URI / absoluteURI
absoluteURI    =  scheme ":" ( hier-part / opaque-part )
hier-part      =  ( net-path / abs-path ) [ "?" query ]
net-path       =  "//" authority [ abs-path ]
abs-path       =  "/" path-segments
```

```
opaque-part    =  uric-no-slash *uric
uric           =  reserved / unreserved / escaped
uric-no-slash  =  unreserved / escaped / ";" / "?" / ":" / "@"
                  / "&" / "=" / "+" / "$" / ","
path-segments  =  segment *( "/" segment )
segment        =  *pchar *( ";" param )
param          =  *pchar
pchar          =  unreserved / escaped /
                  ":" / "@" / "&" / "=" / "+" / "$" / ","
scheme         =  ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )
authority      =  srvr / reg-name
srvr           =  [ [ userinfo "@" ] hostport ]
reg-name       =  1*( unreserved / escaped / "$" / ","
                  / ";" / ":" / "@" / "&" / "=" / "+" )
query          =  *uric
SIP-Version    =  "SIP" "/" 1*DIGIT "." 1*DIGIT

message-header  =  (Accept
                /  Accept-Encoding
                /  Accept-Language
                /  Alert-Info
                /  Allow
                /  Authentication-Info
                /  Authorization
                /  Call-ID
                /  Call-Info
                /  Contact
                /  Content-Disposition
                /  Content-Encoding
                /  Content-Language
                /  Content-Length
                /  Content-Type
                /  CSeq
                /  Date
                /  Error-Info
                /  Expires
                /  From
                /  In-Reply-To
                /  Max-Forwards
                /  MIME-Version
                /  Min-Expires
                /  Organization
                /  Priority
                /  Proxy-Authenticate
                /  Proxy-Authorization
                /  Proxy-Require
                /  Record-Route
                /  Reply-To
```

```
                /  Require
                /  Retry-After
                /  Route
                /  Server
                /  Subject
                /  Supported
                /  Timestamp
                /  To
                /  Unsupported
                /  User-Agent
                /  Via
                /  Warning
                /  WWW-Authenticate
                /  extension-header) CRLF

INVITEm           =  %x49.4E.56.49.54.45 ; INVITE in caps
ACKm              =  %x41.43.4B ; ACK in caps
OPTIONSm          =  %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps
BYEm              =  %x42.59.45 ; BYE in caps
CANCELm           =  %x43.41.4E.43.45.4C ; CANCEL in caps
REGISTERm         =  %x52.45.47.49.53.54.45.52 ; REGISTER in caps
Method            =  INVITEm / ACKm / OPTIONSm / BYEm
                     / CANCELm / REGISTERm
                     / extension-method
extension-method  =  token
Response          =  Status-Line
                     *( message-header )
                     CRLF
                     [ message-body ]

Status-Line     =  SIP-Version SP Status-Code SP Reason-Phrase CRLF
Status-Code     =  Informational
                /  Redirection
                /  Success
                /  Client-Error
                /  Server-Error
                /  Global-Failure
                /  extension-code
extension-code  =  3DIGIT
Reason-Phrase   =  *(reserved / unreserved / escaped
                   / UTF8-NONASCII / UTF8-CONT / SP / HTAB)

Informational  =  "100"  ;  Trying
               /  "180"  ;  Ringing
               /  "181"  ;  Call Is Being Forwarded
               /  "182"  ;  Queued
               /  "183"  ;  Session Progress
```

```
Success  =  "200"  ;  OK

Redirection  =  "300"  ;  Multiple Choices
            /   "301"  ;  Moved Permanently
            /   "302"  ;  Moved Temporarily
            /   "305"  ;  Use Proxy
            /   "380"  ;  Alternative Service

Client-Error  =  "400"  ;  Bad Request
             /   "401"  ;  Unauthorized
             /   "402"  ;  Payment Required
             /   "403"  ;  Forbidden
             /   "404"  ;  Not Found
             /   "405"  ;  Method Not Allowed
             /   "406"  ;  Not Acceptable
             /   "407"  ;  Proxy Authentication Required
             /   "408"  ;  Request Timeout
             /   "410"  ;  Gone
             /   "413"  ;  Request Entity Too Large
             /   "414"  ;  Request-URI Too Large
             /   "415"  ;  Unsupported Media Type
             /   "416"  ;  Unsupported URI Scheme
             /   "420"  ;  Bad Extension
             /   "421"  ;  Extension Required
             /   "423"  ;  Interval Too Brief
             /   "480"  ;  Temporarily not available
             /   "481"  ;  Call Leg/Transaction Does Not Exist
             /   "482"  ;  Loop Detected
             /   "483"  ;  Too Many Hops
             /   "484"  ;  Address Incomplete
             /   "485"  ;  Ambiguous
             /   "486"  ;  Busy Here
             /   "487"  ;  Request Terminated
             /   "488"  ;  Not Acceptable Here
             /   "491"  ;  Request Pending
             /   "493"  ;  Undecipherable

Server-Error  =  "500"  ;  Internal Server Error
             /   "501"  ;  Not Implemented
             /   "502"  ;  Bad Gateway
             /   "503"  ;  Service Unavailable
             /   "504"  ;  Server Time-out
             /   "505"  ;  SIP Version not supported
             /   "513"  ;  Message Too Large
```

```
Global-Failure  =  "600"  ;  Busy Everywhere
               /   "603"  ;  Decline
               /   "604"  ;  Does not exist anywhere
               /   "606"  ;  Not Acceptable

Accept         =  "Accept" HCOLON
                   [ accept-range *(COMMA accept-range) ]
accept-range   =  media-range *(SEMI accept-param)
media-range    =  ( "*/*"
                  / ( m-type SLASH "*" )
                  / ( m-type SLASH m-subtype )
                  ) *( SEMI m-parameter )
accept-param   =  ("q" EQUAL qvalue) / generic-param
qvalue         =  ( "0" [ "." 0*3DIGIT ] )
                  / ( "1" [ "." 0*3("0") ] )
generic-param  =  token [ EQUAL gen-value ]
gen-value      =  token / host / quoted-string

Accept-Encoding  =  "Accept-Encoding" HCOLON
                    [ encoding *(COMMA encoding) ]
encoding         =  codings *(SEMI accept-param)
codings          =  content-coding / "*"
content-coding   =  token

Accept-Language  =  "Accept-Language" HCOLON
                    [ language *(COMMA language) ]
language         =  language-range *(SEMI accept-param)
language-range   =  ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) / "*" )

Alert-Info   =  "Alert-Info" HCOLON alert-param *(COMMA alert-param)
alert-param  =  LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Allow  =  "Allow" HCOLON [Method *(COMMA Method)]

Authorization     =  "Authorization" HCOLON credentials
credentials       =  ("Digest" LWS digest-response)
                     / other-response
digest-response   =  dig-resp *(COMMA dig-resp)
dig-resp          =  username / realm / nonce / digest-uri
                      / dresponse / algorithm / cnonce
                      / opaque / message-qop
                      / nonce-count / auth-param
username          =  "username" EQUAL username-value
username-value    =  quoted-string
digest-uri        =  "uri" EQUAL LDQUOT digest-uri-value RDQUOT
digest-uri-value  =  rquest-uri ; Equal to request-uri as specified
                     by HTTP/1.1
message-qop       =  "qop" EQUAL qop-value
```

```
cnonce            =  "cnonce" EQUAL cnonce-value
cnonce-value      =  nonce-value
nonce-count       =  "nc" EQUAL nc-value
nc-value          =  8LHEX
dresponse         =  "response" EQUAL request-digest
request-digest    =  LDQUOT 32LHEX RDQUOT
auth-param        =  auth-param-name EQUAL
                     ( token / quoted-string )
auth-param-name   =  token
other-response    =  auth-scheme LWS auth-param
                     *(COMMA auth-param)
auth-scheme       =  token

Authentication-Info  =  "Authentication-Info" HCOLON ainfo
                        *(COMMA ainfo)
ainfo                =  nextnonce / message-qop
                         / response-auth / cnonce
                         / nonce-count
nextnonce            =  "nextnonce" EQUAL nonce-value
response-auth        =  "rspauth" EQUAL response-digest
response-digest      =  LDQUOT *LHEX RDQUOT

Call-ID  =  ( "Call-ID" / "i" ) HCOLON callid
callid   =  word [ "@" word ]

Call-Info   =  "Call-Info" HCOLON info *(COMMA info)
info        =  LAQUOT absoluteURI RAQUOT *( SEMI info-param)
info-param  =  ( "purpose" EQUAL ( "icon" / "info"
               / "card" / token ) ) / generic-param

Contact        =  ("Contact" / "m" ) HCOLON
                  ( STAR / (contact-param *(COMMA contact-param)))
contact-param  =  (name-addr / addr-spec) *(SEMI contact-params)
name-addr      =  [ display-name ] LAQUOT addr-spec RAQUOT
addr-spec      =  SIP-URI / SIPS-URI / absoluteURI
display-name   =  *(token LWS)/ quoted-string

contact-params     =  c-p-q / c-p-expires
                      / contact-extension
c-p-q              =  "q" EQUAL qvalue
c-p-expires        =  "expires" EQUAL delta-seconds
contact-extension  =  generic-param
delta-seconds      =  1*DIGIT

Content-Disposition   =  "Content-Disposition" HCOLON
                         disp-type *( SEMI disp-param )
disp-type             =  "render" / "session" / "icon" / "alert"
                         / disp-extension-token
```

```
disp-param            =  handling-param / generic-param
handling-param        =  "handling" EQUAL
                         ( "optional" / "required"
                         / other-handling )
other-handling        =  token
disp-extension-token  =  token

Content-Encoding  =  ( "Content-Encoding" / "e" ) HCOLON
                     content-coding *(COMMA content-coding)

Content-Language  =  "Content-Language" HCOLON
                     language-tag *(COMMA language-tag)
language-tag      =  primary-tag *( "-" subtag )
primary-tag       =  1*8ALPHA
subtag            =  1*8ALPHA

Content-Length  =  ( "Content-Length" / "l" ) HCOLON 1*DIGIT
Content-Type     =  ( "Content-Type" / "c" ) HCOLON media-type
media-type       =  m-type SLASH m-subtype *(SEMI m-parameter)
m-type           =  discrete-type / composite-type
discrete-type    =  "text" / "image" / "audio" / "video"
                    / "application" / extension-token
composite-type   =  "message" / "multipart" / extension-token
extension-token  =  ietf-token / x-token
ietf-token       =  token
x-token          =  "x-" token
m-subtype        =  extension-token / iana-token
iana-token       =  token
m-parameter      =  m-attribute EQUAL m-value
m-attribute      =  token
m-value          =  token / quoted-string

CSeq  =  "CSeq" HCOLON 1*DIGIT LWS Method

Date          =  "Date" HCOLON SIP-date
SIP-date      =  rfc1123-date
rfc1123-date  =  wkday "," SP date1 SP time SP "GMT"
date1         =  2DIGIT SP month SP 4DIGIT
                 ; day month year (e.g., 02 Jun 1982)
time          =  2DIGIT ":" 2DIGIT ":" 2DIGIT
                 ; 00:00:00 - 23:59:59
wkday         =  "Mon" / "Tue" / "Wed"
                 / "Thu" / "Fri" / "Sat" / "Sun"
month         =  "Jan" / "Feb" / "Mar" / "Apr"
                 / "May" / "Jun" / "Jul" / "Aug"
                 / "Sep" / "Oct" / "Nov" / "Dec"

Error-Info  =  "Error-Info" HCOLON error-uri *(COMMA error-uri)
```

```
error-uri   =  LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Expires     =  "Expires" HCOLON delta-seconds
From        =  ( "From" / "f" ) HCOLON from-spec
from-spec   =  ( name-addr / addr-spec )
               *( SEMI from-param )
from-param  =  tag-param / generic-param
tag-param   =  "tag" EQUAL token

In-Reply-To  =  "In-Reply-To" HCOLON callid *(COMMA callid)

Max-Forwards  =  "Max-Forwards" HCOLON 1*DIGIT

MIME-Version  =  "MIME-Version" HCOLON 1*DIGIT "." 1*DIGIT

Min-Expires  =  "Min-Expires" HCOLON delta-seconds

Organization  =  "Organization" HCOLON [TEXT-UTF8-TRIM]

Priority        =  "Priority" HCOLON priority-value
priority-value  =  "emergency" / "urgent" / "normal"
                   / "non-urgent" / other-priority
other-priority  =  token

Proxy-Authenticate  =  "Proxy-Authenticate" HCOLON challenge
challenge           =  ("Digest" LWS digest-cln *(COMMA digest-cln))
                       / other-challenge
other-challenge     =  auth-scheme LWS auth-param
                       *(COMMA auth-param)
digest-cln          =  realm / domain / nonce
                        / opaque / stale / algorithm
                        / qop-options / auth-param
realm               =  "realm" EQUAL realm-value
realm-value         =  quoted-string
domain              =  "domain" EQUAL LDQUOT URI
                       *( 1*SP URI ) RDQUOT
URI                 =  absoluteURI / abs-path
nonce               =  "nonce" EQUAL nonce-value
nonce-value         =  quoted-string
opaque              =  "opaque" EQUAL quoted-string
stale               =  "stale" EQUAL ( "true" / "false" )
algorithm           =  "algorithm" EQUAL ( "MD5" / "MD5-sess"
                       / token )
qop-options         =  "qop" EQUAL LDQUOT qop-value
                       *("," qop-value) RDQUOT
qop-value           =  "auth" / "auth-int" / token

Proxy-Authorization  =  "Proxy-Authorization" HCOLON credentials
```

```
Proxy-Require  =  "Proxy-Require" HCOLON option-tag
                  *(COMMA option-tag)
option-tag     =  token

Record-Route  =  "Record-Route" HCOLON rec-route *(COMMA rec-route)
rec-route     =  name-addr *( SEMI rr-param )
rr-param      =  generic-param

Reply-To      =  "Reply-To" HCOLON rplyto-spec
rplyto-spec   =  ( name-addr / addr-spec )
                 *( SEMI rplyto-param )
rplyto-param  =  generic-param
Require       =  "Require" HCOLON option-tag *(COMMA option-tag)

Retry-After  =  "Retry-After" HCOLON delta-seconds
                [ comment ] *( SEMI retry-param )

retry-param  =  ("duration" EQUAL delta-seconds)
                / generic-param

Route        =  "Route" HCOLON route-param *(COMMA route-param)
route-param  =  name-addr *( SEMI rr-param )

Server           =  "Server" HCOLON server-val *(LWS server-val)
server-val       =  product / comment
product          =  token [SLASH product-version]
product-version  =  token

Subject  =  ( "Subject" / "s" ) HCOLON [TEXT-UTF8-TRIM]

Supported  =  ( "Supported" / "k" ) HCOLON
              [option-tag *(COMMA option-tag)]

Timestamp  =  "Timestamp" HCOLON 1*(DIGIT)
               [ "." *(DIGIT) ] [ LWS delay ]
delay      =  *(DIGIT) [ "." *(DIGIT) ]

To        =  ( "To" / "t" ) HCOLON ( name-addr
             / addr-spec ) *( SEMI to-param )
to-param  =  tag-param / generic-param

Unsupported  =  "Unsupported" HCOLON option-tag *(COMMA option-tag)
User-Agent   =  "User-Agent" HCOLON server-val *(LWS server-val)
```

```
Via               =  ( "Via" / "v" ) HCOLON via-parm *(COMMA via-parm)
via-parm          =  sent-protocol LWS sent-by *( SEMI via-params )
via-params        =  via-ttl / via-maddr
                     / via-received / via-branch
                     / via-extension
via-ttl           =  "ttl" EQUAL ttl
via-maddr         =  "maddr" EQUAL host
via-received      =  "received" EQUAL (IPv4address / IPv6address)
via-branch        =  "branch" EQUAL token
via-extension     =  generic-param
sent-protocol     =  protocol-name SLASH protocol-version
                     SLASH transport
protocol-name     =  "SIP" / token
protocol-version  =  token
transport         =  "UDP" / "TCP" / "TLS" / "SCTP"
                     / other-transport
sent-by           =  host [ COLON port ]
ttl               =  1*3DIGIT ; 0 to 255

Warning        =  "Warning" HCOLON warning-value *(COMMA warning-value)
warning-value  =  warn-code SP warn-agent SP warn-text
warn-code      =  3DIGIT
warn-agent     =  hostport / pseudonym
                  ;  the name or pseudonym of the server adding
                  ;  the Warning header, for use in debugging
warn-text      =  quoted-string
pseudonym      =  token

WWW-Authenticate  =  "WWW-Authenticate" HCOLON challenge

extension-header  =  header-name HCOLON header-value
header-name       =  token
header-value      =  *(TEXT-UTF8char / UTF8-CONT / LWS)
message-body  =  *OCTET
```

## 26 Security Considerations: Threat Model and Security Usage Recommendations

SIP is not an easy protocol to secure. Its use of intermediaries, its multi-faceted trust relationships, its expected usage between elements with no trust at all, and its user-to-user operation make security far from trivial. Security solutions are needed that are deployable today, without extensive coordination, in a wide variety of environments and usages. In order to meet these diverse needs, several distinct mechanisms applicable to different aspects and usages of SIP will be required.

Note that the security of SIP signaling itself has no bearing on the security of protocols used in concert with SIP such as RTP, or with the security implications of any specific bodies SIP might carry (although MIME security plays a substantial role in securing SIP). Any media associated with a session can be encrypted end-to-end independently of any associated SIP signaling. Media encryption is outside the scope of this document.

The considerations that follow first examine a set of classic threat models that broadly identify the security needs of SIP. The set of security services required to address these threats is then detailed, followed by an explanation of several security mechanisms that can be used to provide these services. Next, the requirements for implementers of SIP are enumerated, along with exemplary deployments in which these security mechanisms could be used to improve the security of SIP. Some notes on privacy conclude this section.

### 26.1 Attacks and Threat Models

This section details some threats that should be common to most deployments of SIP. These threats have been chosen specifically to illustrate each of the security services that SIP requires.

The following examples by no means provide an exhaustive list of the threats against SIP; rather, these are "classic" threats that demonstrate the need for particular security services that can potentially prevent whole categories of threats.

These attacks assume an environment in which attackers can potentially read any packet on the network - it is anticipated that SIP will frequently be used on the public Internet. Attackers on the network may be able to modify packets (perhaps at some compromised intermediary). Attackers may wish to steal services, eavesdrop on communications, or disrupt sessions.

#### 26.1.1 Registration Hijacking

The SIP registration mechanism allows a user agent to identify itself to a registrar as a device at which a user (designated by an address of record) is located. A registrar assesses the identity asserted in the From header field of a REGISTER message to determine whether this request can modify the contact addresses associated with the address-of-record in the To header field. While these two fields are frequently the same, there are many valid deployments in which a third-party may register contacts on a user's behalf.

The From header field of a SIP request, however, can be modified arbitrarily by the owner of a UA, and this opens the door to malicious registrations. An attacker that successfully impersonates a party authorized to change contacts associated with an address-of-record could, for example, de-register all existing contacts for a URI and then register their own device as the appropriate contact address, thereby directing all requests for the affected user to the attacker's device.

This threat belongs to a family of threats that rely on the absence of cryptographic assurance of a request's originator. Any SIP UAS that represents a valuable service (a gateway that interworks SIP requests with traditional telephone calls, for example) might want to control access to its resources by authenticating requests that it receives. Even end-user UAs, for example SIP phones, have an interest in ascertaining the identities of originators of requests.

This threat demonstrates the need for security services that enable SIP entities to authenticate the originators of requests.

### 26.1.2 Impersonating a Server

The domain to which a request is destined is generally specified in the Request-URI. UAs commonly contact a server in this domain directly in order to deliver a request. However, there is always a possibility that an attacker could impersonate the remote server, and that the UA's request could be intercepted by some other party.

For example, consider a case in which a redirect server at one domain, chicago.com, impersonates a redirect server at another domain, biloxi.com. A user agent sends a request to biloxi.com, but the redirect server at chicago.com answers with a forged response that has appropriate SIP header fields for a response from biloxi.com. The forged contact addresses in the redirection response could direct the originating UA to inappropriate or insecure resources, or simply prevent requests for biloxi.com from succeeding.

This family of threats has a vast membership, many of which are critical. As a converse to the registration hijacking threat, consider the case in which a registration sent to biloxi.com is intercepted by chicago.com, which replies to the intercepted registration with a forged 301 (Moved Permanently) response. This response might seem to come from biloxi.com yet designate chicago.com as the appropriate registrar. All future REGISTER requests from the originating UA would then go to chicago.com.

Prevention of this threat requires a means by which UAs can authenticate the servers to whom they send requests.

### 26.1.3 Tampering with Message Bodies

As a matter of course, SIP UAs route requests through trusted proxy servers. Regardless of how that trust is established (authentication of proxies is discussed elsewhere in this section), a UA may trust a proxy server to route a request, but not to inspect or possibly modify the bodies contained in that request.

Consider a UA that is using SIP message bodies to communicate session encryption keys for a media session. Although it trusts the proxy server of the domain it is contacting to deliver signaling properly, it may not want the administrators of that domain to be capable of decrypting any subsequent media session. Worse yet, if the proxy server were actively malicious, it could modify the session key, either acting as a man-in-the-middle, or perhaps changing the security characteristics requested by the originating UA.

This family of threats applies not only to session keys, but to most conceivable forms of content carried end-to-end in SIP. These might include MIME bodies that should be rendered to the user, SDP, or encapsulated telephony signals, among others. Attackers might attempt to modify SDP bodies, for example, in order to point RTP media streams to a wiretapping device in order to eavesdrop on subsequent voice communications.

Also note that some header fields in SIP are meaningful end-to-end, for example, Subject. UAs might be protective of these header fields as well as bodies (a malicious intermediary changing the Subject header field might make an important request appear to be spam, for example). However, since many header fields are legitimately inspected or altered by proxy servers as a request is routed, not all header fields should be secured end-to-end.

For these reasons, the UA might want to secure SIP message bodies, and in some limited cases header fields, end-to-end. The security services required for bodies include confidentiality, integrity, and authentication. These end-to-end services should be independent of the means used to secure interactions with intermediaries such as proxy servers.

### 26.1.4 Tearing Down Sessions

Once a dialog has been established by initial messaging, subsequent requests can be sent that modify the state of the dialog and/or session. It is critical that principals in a session can be certain that such requests are not forged by attackers.

Consider a case in which a third-party attacker captures some initial messages in a dialog shared by two parties in order to learn the parameters of the session (To tag, From tag, and so forth) and then inserts a BYE request into the session. The attacker could opt to forge the request such that it seemed to come from either participant. Once the BYE is received by its target, the session will be torn down prematurely.

Similar mid-session threats include the transmission of forged re-INVITEs that alter the session (possibly to reduce session security or redirect media streams as part of a wiretapping attack).

The most effective countermeasure to this threat is the authentication of the sender of the BYE. In this instance, the recipient needs only know that the BYE came from the same party with whom the corresponding dialog was established (as opposed to ascertaining the absolute identity of the sender). Also, if the attacker is unable to learn the parameters of the session due to confidentiality, it would not be possible to forge the BYE. However, some intermediaries (like proxy servers) will need to inspect those parameters as the session is established.

### 26.1.5 Denial of Service and Amplification

Denial-of-service attacks focus on rendering a particular network element unavailable, usually by directing an excessive amount of network traffic at its interfaces. A distributed denial-of-service attack allows one network user to cause multiple network hosts to flood a target host with a large amount of network traffic.

In many architectures, SIP proxy servers face the public Internet in order to accept requests from worldwide IP endpoints. SIP creates a number of potential opportunities for distributed denial-of-service attacks that must be recognized and addressed by the implementers and operators of SIP systems.

Attackers can create bogus requests that contain a falsified source IP address and a corresponding Via header field that identify a targeted host as the originator of the request and then send this request to a large number of SIP network elements, thereby using hapless SIP UAs or proxies to generate denial-of-service traffic aimed at the target.

Similarly, attackers might use falsified Route header field values in a request that identify the target host and then send such messages to forking proxies that will amplify messaging sent to the target.

Record-Route could be used to similar effect when the attacker is certain that the SIP dialog initiated by the request will result in numerous transactions originating in the backwards direction.

A number of denial-of-service attacks open up if REGISTER requests are not properly authenticated and authorized by registrars. Attackers could de-register some or all users in an administrative domain, thereby preventing these users from being invited to new sessions. An attacker could also register a large number of contacts designating the same host for a given address-of-record in order to use the registrar and any associated proxy servers as amplifiers in a denial-of-service attack. Attackers might also attempt to deplete available memory and disk resources of a registrar by registering huge numbers of bindings.

The use of multicast to transmit SIP requests can greatly increase the potential for denial-of-service attacks.

These problems demonstrate a general need to define architectures that minimize the risks of denial-of-service, and the need to be mindful in recommendations for security mechanisms of this class of attacks.

### 26.2 Security Mechanisms

From the threats described above, we gather that the fundamental security services required for the SIP protocol are: preserving the confidentiality and integrity of messaging, preventing replay attacks or message spoofing, providing for the authentication and privacy of the participants in a session, and preventing denial-of-service attacks. Bodies within SIP messages separately require the security services of confidentiality, integrity, and authentication.

Rather than defining new security mechanisms specific to SIP, SIP reuses wherever possible existing security models derived from the HTTP and SMTP space.

Full encryption of messages provides the best means to preserve the confidentiality of signaling - it can also guarantee that messages are not modified by any malicious intermediaries. However, SIP requests and responses cannot be naively encrypted end-to-end in their entirety because message fields such as the Request-URI, Route, and Via need to be visible to proxies in most network architectures so that SIP requests are routed correctly. Note that proxy servers need to modify some features of messages as well (such as adding Via header field values) in order for SIP to function. Proxy servers must therefore be trusted, to some degree, by SIP UAs. To this purpose, low-layer security mechanisms for SIP are recommended, which

encrypt the entire SIP requests or responses on the wire on a hop-by-hop basis, and that allow endpoints to verify the identity of proxy servers to whom they send requests.

SIP entities also have a need to identify one another in a secure fashion. When a SIP endpoint asserts the identity of its user to a peer UA or to a proxy server, that identity should in some way be verifiable. A cryptographic authentication mechanism is provided in SIP to address this requirement.

An independent security mechanism for SIP message bodies supplies an alternative means of end-to-end mutual authentication, as well as providing a limit on the degree to which user agents must trust intermediaries.

### 26.2.1 Transport and Network Layer Security

Transport or network layer security encrypts signaling traffic, guaranteeing message confidentiality and integrity.

Oftentimes, certificates are used in the establishment of lower-layer security, and these certificates can also be used to provide a means of authentication in many architectures.

Two popular alternatives for providing security at the transport and network layer are, respectively, TLS [25] and IPSec [26].

IPSec is a set of network-layer protocol tools that collectively can be used as a secure replacement for traditional IP (Internet Protocol). IPSec is most commonly used in architectures in which a set of hosts or administrative domains have an existing trust relationship with one another. IPSec is usually implemented at the operating system level in a host, or on a security gateway that provides confidentiality and integrity for all traffic it receives from a particular interface (as in a VPN architecture). IPSec can also be used on a hop-by-hop basis.

In many architectures IPSec does not require integration with SIP applications; IPSec is perhaps best suited to deployments in which adding security directly to SIP hosts would be arduous. UAs that have a pre-shared keying relationship with their first-hop proxy server are also good candidates to use IPSec. Any deployment of IPSec for SIP would require an IPSec profile describing the protocol tools that would be required to secure SIP. No such profile is given in this document.

TLS provides transport-layer security over connection-oriented protocols (for the purposes of this document, TCP); "tls" (signifying TLS over TCP) can be specified as the desired transport protocol within a Via header field value or a SIP-URI. TLS is most suited to architectures in which hop-by-hop security is required between hosts with no pre-existing trust association. For example, Alice trusts her local proxy server, which after a certificate exchange decides to trust Bob's local proxy server, which Bob trusts, hence Bob and Alice can communicate securely.

TLS must be tightly coupled with a SIP application. Note that transport mechanisms are specified on a hop-by-hop basis in SIP, thus a UA that sends requests over TLS to a proxy server has no assurance that TLS will be used end-to-end.

The TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite [6] MUST be supported at a minimum by implementers when TLS is used in a SIP application. For purposes of backwards compatibility, proxy servers, redirect servers, and registrars SHOULD support TLS_RSA_WITH_3DES_EDE_CBC_SHA. Implementers MAY also support any other ciphersuite.

### 26.2.2 SIPS URI Scheme

The SIPS URI scheme adheres to the syntax of the SIP URI (described in 19), although the scheme string is "sips" rather than "sip". The semantics of SIPS are very different from the SIP URI, however. SIPS allows resources to specify that they should be reached securely.

A SIPS URI can be used as an address-of-record for a particular user - the URI by which the user is canonically known (on their business cards, in the From header field of their requests, in the To header field of REGISTER requests). When used as the Request-URI of a request, the SIPS scheme signifies that each hop over which the request is forwarded, until the request reaches the SIP entity responsible for the domain portion of the Request-URI, must be secured with TLS; once it reaches the domain in question it is handled in accordance with local security and routing policy, quite possibly using TLS for any last hop to a UAS. When used by the originator of a request (as would be the case if they employed a SIPS URI as the address-of-record of the target), SIPS dictates that the entire request path to the target domain be so secured.

The SIPS scheme is applicable to many of the other ways in which SIP URIs are used in SIP today in addition to the Request-URI, including in addresses-of-record, contact addresses (the contents of Contact headers, including those of REGISTER methods), and Route headers. In each instance, the SIPS URI scheme allows these existing fields to

designate secure resources. The manner in which a SIPS URI is dereferenced in any of these contexts has its own security properties which are detailed in [4].

The use of SIPS in particular entails that mutual TLS authentication SHOULD be employed, as SHOULD the ciphersuite TLS_RSA_WITH_AES_128_CBC_SHA. Certificates received in the authentication process SHOULD be validated with root certificates held by the client; failure to validate a certificate SHOULD result in the failure of the request.

> Note that in the SIPS URI scheme, transport is independent of TLS, and thus "sips:alice@atlanta.com;transport=tcp" and "sips:alice@atlanta.com;transport=sctp" are both valid (although note that UDP is not a valid transport for SIPS). The use of "transport=tls" has consequently been deprecated, partly because it was specific to a single hop of the request. This is a change since RFC 2543.

Users that distribute a SIPS URI as an address-of-record may elect to operate devices that refuse requests over insecure transports.

### 26.2.3 HTTP Authentication

SIP provides a challenge capability, based on HTTP authentication, that relies on the 401 and 407 response codes as well as header fields for carrying challenges and credentials. Without significant modification, the reuse of the HTTP Digest authentication scheme in SIP allows for replay protection and one-way authentication.

The usage of Digest authentication in SIP is detailed in Section 22.

### 26.2.4 S/MIME

As is discussed above, encrypting entire SIP messages end-to-end for the purpose of confidentiality is not appropriate because network intermediaries (like proxy servers) need to view certain header fields in order to route messages correctly, and if these intermediaries are excluded from security associations, then SIP messages will essentially be non-routable.

However, S/MIME allows SIP UAs to encrypt MIME bodies within SIP, securing these bodies end-to-end without affecting message headers. S/MIME can provide end-to-end confidentiality and integrity for message bodies, as well as mutual authentication. It is also possible to use S/MIME to provide a form of integrity and confidentiality for SIP header fields through SIP message tunneling.

The usage of S/MIME in SIP is detailed in Section 23.

### 26.3 Implementing Security Mechanisms

#### 26.3.1 Requirements for Implementers of SIP

Proxy servers, redirect servers, and registrars MUST implement TLS, and MUST support both mutual and one-way authentication. It is strongly RECOMMENDED that UAs be capable initiating TLS; UAs MAY also be capable of acting as a TLS server. Proxy servers, redirect servers, and registrars SHOULD possess a site certificate whose subject corresponds to their canonical hostname. UAs MAY have certificates of their own for mutual authentication with TLS, but no provisions are set forth in this document for their use. All SIP elements that support TLS MUST have a mechanism for validating certificates received during TLS negotiation; this entails possession of one or more root certificates issued by certificate authorities (preferably well-known distributors of site certificates comparable to those that issue root certificates for web browsers).

All SIP elements that support TLS MUST also support the SIPS URI scheme.

Proxy servers, redirect servers, registrars, and UAs MAY also implement IPSec or other lower-layer security protocols.

When a UA attempts to contact a proxy server, redirect server, or registrar, the UAC SHOULD initiate a TLS connection over which it will send SIP messages. In some architectures, UASs MAY receive requests over such TLS connections as well.

Proxy servers, redirect servers, registrars, and UAs MUST implement Digest Authorization, encompassing all of the aspects required in 22. Proxy servers, redirect servers, and registrars SHOULD be configured with at least one Digest realm, and at least one "realm" string supported by a given server SHOULD correspond to the server's hostname or domainname.

UAs MAY support the signing and encrypting of MIME bodies, and transference of credentials with S/MIME as described in Section 23. If a UA holds one or more root certificates of certificate authorities in order to validate certificates for TLS or IPSec, it SHOULD be capable of reusing these to verify S/MIME certificates, as appropriate. A UA MAY hold root certificates specifically for validating S/MIME certificates.

Note that is it anticipated that future security extensions may upgrade the normative strength associated with S/MIME as S/MIME implementations appear and the problem space becomes better understood.

### 26.3.2 Security Solutions

The operation of these security mechanisms in concert can follow the existing web and email security models to some degree. At a high level, UAs authenticate themselves to servers (proxy servers, redirect servers, and registrars) with a Digest username and password; servers authenticate themselves to UAs one hop away, or to another server one hop away (and vice versa), with a site certificate delivered by TLS.

On a peer-to-peer level, UAs trust the network to authenticate one another ordinarily; however, S/MIME can also be used to provide direct authentication when the network does not, or if the network itself is not trusted.

The following is an illustrative example in which these security mechanisms are used by various UAs and servers to prevent the sorts of threats described in Section 26.1. While implementers and network administrators MAY follow the normative guidelines given in the remainder of this section, these are provided only as example implementations.

#### 26.3.2.1 Registration

When a UA comes online and registers with its local administrative domain, it SHOULD establish a TLS connection with its registrar (Section 10 describes how the UA reaches its registrar). The registrar SHOULD offer a certificate to the UA, and the site identified by the certificate MUST correspond with the domain in which the UA intends to register; for example, if the UA intends to register the address-of-record 'alice@atlanta.com', the site certificate must identify a host within the atlanta.com domain (such as sip.atlanta.com). When it receives the TLS Certificate message, the UA SHOULD verify the certificate and inspect the site identified by the certificate. If the certificate is invalid, revoked, or if it does not identify the appropriate party, the UA MUST NOT send the REGISTER message and otherwise proceed with the registration.

When a valid certificate has been provided by the registrar, the UA knows that the registrar is not an attacker who might redirect the UA, steal passwords, or attempt any similar attacks.

The UA then creates a REGISTER request that SHOULD be addressed to a Request-URI corresponding to the site certificate received from the registrar. When the UA sends the REGISTER request over the existing TLS connection, the registrar SHOULD challenge the request with a 401 (Proxy Authentication Required) response. The "realm" parameter within the Proxy-Authenticate header field of the response SHOULD correspond to the domain previously given by the site certificate. When the UAC receives the challenge, it SHOULD either prompt the user for credentials or take an appropriate credential from a keyring corresponding to the "realm" parameter in the challenge. The username of this credential SHOULD correspond with the "userinfo" portion of the URI in the To header field of the REGISTER request. Once the Digest credentials have been inserted into an appropriate Proxy-Authorization header field, the REGISTER should be resubmitted to the registrar.

> Since the registrar requires the user agent to authenticate itself, it would be difficult for an attacker to forge REGISTER requests for the user's address-of-record. Also note that since the REGISTER is sent over a confidential TLS connection, attackers will not be able to intercept the REGISTER to record credentials for any possible replay attack.

Once the registration has been accepted by the registrar, the UA SHOULD leave this TLS connection open provided that the registrar also acts as the proxy server to which requests are sent for users in this administrative domain. The existing TLS connection will be reused to deliver incoming requests to the UA that has just completed registration.

> Because the UA has already authenticated the server on the other side of the TLS connection, all requests that come over this connection are known to have passed through the proxy server - attackers cannot create spoofed requests that appear to have been sent through that proxy server.

#### 26.3.2.2 Interdomain Requests

Now let's say that Alice's UA would like to initiate a session with a user in a remote administrative domain, namely "bob@biloxi.com". We will also say that the local administrative domain (atlanta.com) has a local outbound proxy.

The proxy server that handles inbound requests for an administrative domain MAY also act as a local outbound proxy; for simplicity's sake we'll assume this to be the case for atlanta.com (otherwise the user agent would initiate a new TLS connection to a separate server at this point). Assuming that the client has completed the registration

process described in the preceding section, it SHOULD reuse the TLS connection to the local proxy server when it sends an INVITE request to another user. The UA SHOULD reuse cached credentials in the INVITE to avoid prompting the user unnecessarily.

When the local outbound proxy server has validated the credentials presented by the UA in the INVITE, it SHOULD inspect the Request-URI to determine how the message should be routed (see [4]). If the "domainname" portion of the Request-URI had corresponded to the local domain (atlanta.com) rather than biloxi.com, then the proxy server would have consulted its location service to determine how best to reach the requested user.

> Had "alice@atlanta.com" been attempting to contact, say, "alex@atlanta.com", the local proxy would have proxied to the request to the TLS connection Alex had established with the registrar when he registered. Since Alex would receive this request over his authenticated channel, he would be assured that Alice's request had been authorized by the proxy server of the local administrative domain.

However, in this instance the Request-URI designates a remote domain. The local outbound proxy server at atlanta.com SHOULD therefore establish a TLS connection with the remote proxy server at biloxi.com. Since both of the participants in this TLS connection are servers that possess site certificates, mutual TLS authentication SHOULD occur. Each side of the connection SHOULD verify and inspect the certificate of the other, noting the domain name that appears in the certificate for comparison with the header fields of SIP messages. The atlanta.com proxy server, for example, SHOULD verify at this stage that the certificate received from the remote side corresponds with the biloxi.com domain. Once it has done so, and TLS negotiation has completed, resulting in a secure channel between the two proxies, the atlanta.com proxy can forward the INVITE request to biloxi.com.

The proxy server at biloxi.com SHOULD inspect the certificate of the proxy server at atlanta.com in turn and compare the domain asserted by the certificate with the "domainname" portion of the From header field in the INVITE request. The biloxi proxy MAY have a strict security policy that requires it to reject requests that do not match the administrative domain from which they have been proxied.

> Such security policies could be instituted to prevent the SIP equivalent of SMTP 'open relays' that are frequently exploited to generate spam.

This policy, however, only guarantees that the request came from the domain it ascribes to itself; it does not allow biloxi.com to ascertain how atlanta.com authenticated Alice. Only if biloxi.com has some other way of knowing atlanta.com's authentication policies could it possibly ascertain how Alice proved her identity. biloxi.com might then institute an even stricter policy that forbids requests that come from domains that are not known administratively to share a common authentication policy with biloxi.com.

Once the INVITE has been approved by the biloxi proxy, the proxy server SHOULD identify the existing TLS channel, if any, associated with the user targeted by this request (in this case "bob@biloxi.com"). The INVITE should be proxied through this channel to Bob. Since the request is received over a TLS connection that had previously been authenticated as the biloxi proxy, Bob knows that the From header field was not tampered with and that atlanta.com has validated Alice, although not necessarily whether or not to trust Alice's identity.

Before they forward the request, both proxy servers SHOULD add a Record-Route header field to the request so that all future requests in this dialog will pass through the proxy servers. The proxy servers can thereby continue to provide security services for the lifetime of this dialog. If the proxy servers do not add themselves to the Record-Route, future messages will pass directly end-to-end between Alice and Bob without any security services (unless the two parties agree on some independent end-to-end security such as S/MIME). In this respect the SIP trapezoid model can provide a nice structure where conventions of agreement between the site proxies can provide a reasonably secure channel between Alice and Bob.

> An attacker preying on this architecture would, for example, be unable to forge a BYE request and insert it into the signaling stream between Bob and Alice because the attacker has no way of ascertaining the parameters of the session and also because the integrity mechanism transitively protects the traffic between Alice and Bob.

#### 26.3.2.3 Peer-to-Peer Requests

Alternatively, consider a UA asserting the identity "carol@chicago.com" that has no local outbound proxy. When Carol wishes to send an INVITE to "bob@biloxi.com", her UA SHOULD initiate a TLS connection with the biloxi proxy directly (using the mechanism described in [4] to determine how to best to reach the given Request-URI). When her UA receives a certificate from the biloxi proxy, it SHOULD be verified normally before she passes her INVITE across the TLS connection. However, Carol has no means of proving

her identity to the biloxi proxy, but she does have a CMS-detached signature over a "message/sip" body in the INVITE. It is unlikely in this instance that Carol would have any credentials in the biloxi.com realm, since she has no formal association with biloxi.com. The biloxi proxy MAY also have a strict policy that precludes it from even bothering to challenge requests that do not have biloxi.com in the "domainname" portion of the From header field - it treats these users as unauthenticated.

The biloxi proxy has a policy for Bob that all non-authenticated requests should be redirected to the appropriate contact address registered against 'bob@biloxi.com', namely <sip:bob@192.0.2.4>. Carol receives the redirection response over the TLS connection she established with the biloxi proxy, so she trusts the veracity of the contact address.

Carol SHOULD then establish a TCP connection with the designated address and send a new INVITE with a Request-URI containing the received contact address (recomputing the signature in the body as the request is readied). Bob receives this INVITE on an insecure interface, but his UA inspects and, in this instance, recognizes the From header field of the request and subsequently matches a locally cached certificate with the one presented in the signature of the body of the INVITE. He replies in similar fashion, authenticating himself to Carol, and a secure dialog begins.

> Sometimes firewalls or NATs in an administrative domain could preclude the establishment of a direct TCP connection to a UA. In these cases, proxy servers could also potentially relay requests to UAs in a way that has no trust implications (for example, forgoing an existing TLS connection and forwarding the request over cleartext TCP) as local policy dictates.

#### 26.3.2.4 DoS Protection

In order to minimize the risk of a denial-of-service attack against architectures using these security solutions, implementers should take note of the following guidelines.

When the host on which a SIP proxy server is operating is routable from the public Internet, it SHOULD be deployed in an administrative domain with defensive operational policies (blocking source-routed traffic, preferably filtering ping traffic). Both TLS and IPSec can also make use of bastion hosts at the edges of administrative domains that participate in the security associations to aggregate secure tunnels and sockets. These bastion hosts can also take the brunt of denial-of-service attacks, ensuring that SIP hosts within the administrative domain are not encumbered with superfluous messaging.

No matter what security solutions are deployed, floods of messages directed at proxy servers can lock up proxy server resources and prevent desirable traffic from reaching its destination. There is a computational expense associated with processing a SIP transaction at a proxy server, and that expense is greater for stateful proxy servers than it is for stateless proxy servers. Therefore, stateful proxies are more susceptible to flooding than stateless proxy servers.

UAs and proxy servers SHOULD challenge questionable requests with only a single 401 (Unauthorized) or 407 (Proxy Authentication Required), forgoing the normal response retransmission algorithm, and thus behaving statelessly towards unauthenticated requests.

> Retransmitting the 401 (Unauthorized) or 407 (Proxy Authentication Required) status response amplifies the problem of an attacker using a falsified header field value (such as Via) to direct traffic to a third party.

In summary, the mutual authentication of proxy servers through mechanisms such as TLS significantly reduces the potential for rogue intermediaries to introduce falsified requests or responses that can deny service. This commensurately makes it harder for attackers to make innocent SIP nodes into agents of amplification.

### 26.4 Limitations

Although these security mechanisms, when applied in a judicious manner, can thwart many threats, there are limitations in the scope of the mechanisms that must be understood by implementers and network operators.

#### 26.4.1 HTTP Digest

One of the primary limitations of using HTTP Digest in SIP is that the integrity mechanisms in Digest do not work very well for SIP. Specifically, they offer protection of the Request-URI and the method of a message, but not for any of the header fields that UAs would most likely wish to secure.

The existing replay protection mechanisms described in RFC 2617 also have some limitations for SIP. The next-nonce mechanism, for example, does not support pipelined requests. The nonce-count mechanism should be used for replay protection.

Another limitation of HTTP Digest is the scope of realms. Digest is valuable when a user wants to authenticate themselves to a resource with which they have a pre-existing association, like a service

provider of which the user is a customer (which is quite a common scenario and thus Digest provides an extremely useful function). By way of contrast, the scope of TLS is interdomain or multirealm, since certificates are often globally verifiable, so that the UA can authenticate the server with no pre-existing association.

### 26.4.2 S/MIME

The largest outstanding defect with the S/MIME mechanism is the lack of a prevalent public key infrastructure for end users. If self-signed certificates (or certificates that cannot be verified by one of the participants in a dialog) are used, the SIP-based key exchange mechanism described in Section 23.2 is susceptible to a man-in-the-middle attack with which an attacker can potentially inspect and modify S/MIME bodies. The attacker needs to intercept the first exchange of keys between the two parties in a dialog, remove the existing CMS-detached signatures from the request and response, and insert a different CMS-detached signature containing a certificate supplied by the attacker (but which seems to be a certificate for the proper address-of-record). Each party will think they have exchanged keys with the other, when in fact each has the public key of the attacker.

It is important to note that the attacker can only leverage this vulnerability on the first exchange of keys between two parties - on subsequent occasions, the alteration of the key would be noticeable to the UAs. It would also be difficult for the attacker to remain in the path of all future dialogs between the two parties over time (as potentially days, weeks, or years pass).

SSH is susceptible to the same man-in-the-middle attack on the first exchange of keys; however, it is widely acknowledged that while SSH is not perfect, it does improve the security of connections. The use of key fingerprints could provide some assistance to SIP, just as it does for SSH. For example, if two parties use SIP to establish a voice communications session, each could read off the fingerprint of the key they received from the other, which could be compared against the original. It would certainly be more difficult for the man-in-the-middle to emulate the voices of the participants than their signaling (a practice that was used with the Clipper chip-based secure telephone).

The S/MIME mechanism allows UAs to send encrypted requests without preamble if they possess a certificate for the destination address-of-record on their keyring. However, it is possible that any particular device registered for an address-of-record will not hold the certificate that has been previously employed by the device's current user, and that it will therefore be unable to process an

encrypted request properly, which could lead to some avoidable error signaling. This is especially likely when an encrypted request is forked.

The keys associated with S/MIME are most useful when associated with a particular user (an address-of-record) rather than a device (a UA). When users move between devices, it may be difficult to transport private keys securely between UAs; how such keys might be acquired by a device is outside the scope of this document.

Another, more prosaic difficulty with the S/MIME mechanism is that it can result in very large messages, especially when the SIP tunneling mechanism described in Section 23.4 is used. For that reason, it is RECOMMENDED that TCP should be used as a transport protocol when S/MIME tunneling is employed.

### 26.4.3 TLS

The most commonly voiced concern about TLS is that it cannot run over UDP; TLS requires a connection-oriented underlying transport protocol, which for the purposes of this document means TCP.

It may also be arduous for a local outbound proxy server and/or registrar to maintain many simultaneous long-lived TLS connections with numerous UAs. This introduces some valid scalability concerns, especially for intensive ciphersuites. Maintaining redundancy of long-lived TLS connections, especially when a UA is solely responsible for their establishment, could also be cumbersome.

TLS only allows SIP entities to authenticate servers to which they are adjacent; TLS offers strictly hop-by-hop security. Neither TLS, nor any other mechanism specified in this document, allows clients to authenticate proxy servers to whom they cannot form a direct TCP connection.

### 26.4.4 SIPS URIs

Actually using TLS on every segment of a request path entails that the terminating UAS must be reachable over TLS (perhaps registering with a SIPS URI as a contact address). This is the preferred use of SIPS. Many valid architectures, however, use TLS to secure part of the request path, but rely on some other mechanism for the final hop to a UAS, for example. Thus SIPS cannot guarantee that TLS usage will be truly end-to-end. Note that since many UAs will not accept incoming TLS connections, even those UAs that do support TLS may be required to maintain persistent TLS connections as described in the TLS limitations section above in order to receive requests over TLS as a UAS.

Location services are not required to provide a SIPS binding for a SIPS Request-URI. Although location services are commonly populated by user registrations (as described in Section 10.2.1), various other protocols and interfaces could conceivably supply contact addresses for an AOR, and these tools are free to map SIPS URIs to SIP URIs as appropriate. When queried for bindings, a location service returns its contact addresses without regard for whether it received a request with a SIPS Request-URI. If a redirect server is accessing the location service, it is up to the entity that processes the Contact header field of a redirection to determine the propriety of the contact addresses.

Ensuring that TLS will be used for all of the request segments up to the target domain is somewhat complex. It is possible that cryptographically authenticated proxy servers along the way that are non-compliant or compromised may choose to disregard the forwarding rules associated with SIPS (and the general forwarding rules in Section 16.6). Such malicious intermediaries could, for example, retarget a request from a SIPS URI to a SIP URI in an attempt to downgrade security.

Alternatively, an intermediary might legitimately retarget a request from a SIP to a SIPS URI. Recipients of a request whose Request-URI uses the SIPS URI scheme thus cannot assume on the basis of the Request-URI alone that SIPS was used for the entire request path (from the client onwards).

To address these concerns, it is RECOMMENDED that recipients of a request whose Request-URI contains a SIP or SIPS URI inspect the To header field value to see if it contains a SIPS URI (though note that it does not constitute a breach of security if this URI has the same scheme but is not equivalent to the URI in the To header field). Although clients may choose to populate the Request-URI and To header field of a request differently, when SIPS is used this disparity could be interpreted as a possible security violation, and the request could consequently be rejected by its recipient. Recipients MAY also inspect the Via header chain in order to double-check whether or not TLS was used for the entire request path until the local administrative domain was reached. S/MIME may also be used by the originating UAC to help ensure that the original form of the To header field is carried end-to-end.

If the UAS has reason to believe that the scheme of the Request-URI has been improperly modified in transit, the UA SHOULD notify its user of a potential security breach.

As a further measure to prevent downgrade attacks, entities that accept only SIPS requests MAY also refuse connections on insecure ports.

End users will undoubtedly discern the difference between SIPS and SIP URIs, and they may manually edit them in response to stimuli. This can either benefit or degrade security. For example, if an attacker corrupts a DNS cache, inserting a fake record set that effectively removes all SIPS records for a proxy server, then any SIPS requests that traverse this proxy server may fail. When a user, however, sees that repeated calls to a SIPS AOR are failing, they could on some devices manually convert the scheme from SIPS to SIP and retry. Of course, there are some safeguards against this (if the destination UA is truly paranoid it could refuse all non-SIPS requests), but it is a limitation worth noting. On the bright side, users might also divine that 'SIPS' would be valid even when they are presented only with a SIP URI.

### 26.5 Privacy

SIP messages frequently contain sensitive information about their senders - not just what they have to say, but with whom they communicate, when they communicate and for how long, and from where they participate in sessions. Many applications and their users require that this sort of private information be hidden from any parties that do not need to know it.

Note that there are also less direct ways in which private information can be divulged. If a user or service chooses to be reachable at an address that is guessable from the person's name and organizational affiliation (which describes most addresses-of-record), the traditional method of ensuring privacy by having an unlisted "phone number" is compromised. A user location service can infringe on the privacy of the recipient of a session invitation by divulging their specific whereabouts to the caller; an implementation consequently SHOULD be able to restrict, on a per-user basis, what kind of location and availability information is given out to certain classes of callers. This is a whole class of problem that is expected to be studied further in ongoing SIP work.

In some cases, users may want to conceal personal information in header fields that convey identity. This can apply not only to the From and related headers representing the originator of the request, but also the To - it may not be appropriate to convey to the final destination a speed-dialing nickname, or an unexpanded identifier for a group of targets, either of which would be removed from the Request-URI as the request is routed, but not changed in the To

header field if the two were initially identical. Thus it MAY be desirable for privacy reasons to create a To header field that differs from the Request-URI.

## 27 IANA Considerations

All method names, header field names, status codes, and option tags used in SIP applications are registered with IANA through instructions in an IANA Considerations section in an RFC.

The specification instructs the IANA to create four new sub-registries under http://www.iana.org/assignments/sip-parameters: Option Tags, Warning Codes (warn-codes), Methods and Response Codes, added to the sub-registry of Header Fields that is already present there.

### 27.1 Option Tags

This specification establishes the Option Tags sub-registry under http://www.iana.org/assignments/sip-parameters.

Option tags are used in header fields such as Require, Supported, Proxy-Require, and Unsupported in support of SIP compatibility mechanisms for extensions (Section 19.2). The option tag itself is a string that is associated with a particular SIP option (that is, an extension). It identifies the option to SIP endpoints.

Option tags are registered by the IANA when they are published in standards track RFCs. The IANA Considerations section of the RFC must include the following information, which appears in the IANA registry along with the RFC number of the publication.

- Name of the option tag. The name MAY be of any length, but SHOULD be no more than twenty characters long. The name MUST consist of alphanum (Section 25) characters only.

- Descriptive text that describes the extension.

### 27.2 Warn-Codes

This specification establishes the Warn-codes sub-registry under http://www.iana.org/assignments/sip-parameters and initiates its population with the warn-codes listed in Section 20.43. Additional warn-codes are registered by RFC publication.

The descriptive text for the table of warn-codes is:

Warning codes provide information supplemental to the status code in SIP response messages when the failure of the transaction results from a Session Description Protocol (SDP) (RFC 2327 [1]) problem.

The "warn-code" consists of three digits. A first digit of "3" indicates warnings specific to SIP. Until a future specification describes uses of warn-codes other than 3xx, only 3xx warn-codes may be registered.

Warnings 300 through 329 are reserved for indicating problems with keywords in the session description, 330 through 339 are warnings related to basic network services requested in the session description, 370 through 379 are warnings related to quantitative QoS parameters requested in the session description, and 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

### 27.3 Header Field Names

This obsoletes the IANA instructions about the header sub-registry under http://www.iana.org/assignments/sip-parameters.

The following information needs to be provided in an RFC publication in order to register a new header field name:

- The RFC number in which the header is registered;
- the name of the header field being registered;
- a compact form version for that header field, if one is defined;

Some common and widely used header fields MAY be assigned one-letter compact forms (Section 7.3.3). Compact forms can only be assigned after SIP working group review, followed by RFC publication.

### 27.4 Method and Response Codes

This specification establishes the Method and Response-Code sub-registries under http://www.iana.org/assignments/sip-parameters and initiates their population as follows. The initial Methods table is:

```
      INVITE                  [RFC3261]
      ACK                     [RFC3261]
      BYE                     [RFC3261]
      CANCEL                  [RFC3261]
      REGISTER                [RFC3261]
      OPTIONS                 [RFC3261]
      INFO                    [RFC2976]
```

The response code table is initially populated from Section 21, the portions labeled Informational, Success, Redirection, Client-Error, Server-Error, and Global-Failure. The table has the following format:

```
   Type (e.g., Informational)
         Number    Default Reason Phrase         [RFC3261]
```

The following information needs to be provided in an RFC publication in order to register a new response code or method:

- The RFC number in which the method or response code is registered;
- the number of the response code or name of the method being registered;
- the default reason phrase for that response code, if applicable;

### 27.5 The "message/sip" MIME type.

This document registers the "message/sip" MIME media type in order to allow SIP messages to be tunneled as bodies within SIP, primarily for end-to-end security purposes. This media type is defined by the following information:

```
   Media type name: message
   Media subtype name: sip
   Required parameters: none

   Optional parameters: version
      version: The SIP-Version number of the enclosed message (e.g.,
      "2.0"). If not present, the version defaults to "2.0".
   Encoding scheme: SIP messages consist of an 8-bit header
      optionally followed by a binary MIME data object. As such, SIP
      messages must be treated as binary. Under normal circumstances
      SIP messages are transported over binary-capable transports, no
      special encodings are needed.
```

```
Security considerations: see below
   Motivation and examples of this usage as a security mechanism
   in concert with S/MIME are given in 23.4.
```

### 27.6 New Content-Disposition Parameter Registrations

This document also registers four new Content-Disposition header "disposition-types": alert, icon, session and render. The authors request that these values be recorded in the IANA registry for Content-Dispositions.

Descriptions of these "disposition-types", including motivation and examples, are given in Section 20.11.

Short descriptions suitable for the IANA registry are:

| | |
|---|---|
| alert | the body is a custom ring tone to alert the user |
| icon | the body is displayed as an icon to the user |
| render | the body should be displayed to the user |
| session | the body describes a communications session, for example, as RFC 2327 SDP body |

## 28 Changes From RFC 2543

This RFC revises RFC 2543. It is mostly backwards compatible with RFC 2543. The changes described here fix many errors discovered in RFC 2543 and provide information on scenarios not detailed in RFC 2543. The protocol has been presented in a more cleanly layered model here.

We break the differences into functional behavior that is a substantial change from RFC 2543, which has impact on interoperability or correct operation in some cases, and functional behavior that is different from RFC 2543 but not a potential source of interoperability problems. There have been countless clarifications as well, which are not documented here.

### 28.1 Major Functional Changes

- When a UAC wishes to terminate a call before it has been answered, it sends CANCEL. If the original INVITE still returns a 2xx, the UAC then sends BYE. BYE can only be sent on an existing call leg (now called a dialog in this RFC), whereas it could be sent at any time in RFC 2543.

- The SIP BNF was converted to be RFC 2234 compliant.

- SIP URL BNF was made more general, allowing a greater set of characters in the user part. Furthermore, comparison rules were simplified to be primarily case-insensitive, and detailed handling of comparison in the presence of parameters was described. The most substantial change is that a URI with a parameter with the default value does not match a URI without that parameter.

- Removed Via hiding. It had serious trust issues, since it relied on the next hop to perform the obfuscation process. Instead, Via hiding can be done as a local implementation choice in stateful proxies, and thus is no longer documented.

- In RFC 2543, CANCEL and INVITE transactions were intermingled. They are separated now. When a user sends an INVITE and then a CANCEL, the INVITE transaction still terminates normally. A UAS needs to respond to the original INVITE request with a 487 response.

- Similarly, CANCEL and BYE transactions were intermingled; RFC 2543 allowed the UAS not to send a response to INVITE when a BYE was received. That is disallowed here. The original INVITE needs a response.

- In RFC 2543, UAs needed to support only UDP. In this RFC, UAs need to support both UDP and TCP.

- In RFC 2543, a forking proxy only passed up one challenge from downstream elements in the event of multiple challenges. In this RFC, proxies are supposed to collect all challenges and place them into the forwarded response.

- In Digest credentials, the URI needs to be quoted; this is unclear from RFC 2617 and RFC 2069 which are both inconsistent on it.

- SDP processing has been split off into a separate specification [13], and more fully specified as a formal offer/answer exchange process that is effectively tunneled through SIP. SDP is allowed in INVITE/200 or 200/ACK for baseline SIP implementations; RFC 2543 alluded to the ability to use it in INVITE, 200, and ACK in a single transaction, but this was not well specified. More complex SDP usages are allowed in extensions.

- Added full support for IPv6 in URIs and in the Via header field. Support for IPv6 in Via has required that its header field parameters allow the square bracket and colon characters. These characters were previously not permitted. In theory, this could cause interop problems with older implementations. However, we have observed that most implementations accept any non-control ASCII character in these parameters.

- DNS SRV procedure is now documented in a separate specification [4]. This procedure uses both SRV and NAPTR resource records and no longer combines data from across SRV records as described in RFC 2543.

- Loop detection has been made optional, supplanted by a mandatory usage of Max-Forwards. The loop detection procedure in RFC 2543 had a serious bug which would report "spirals" as an error condition when it was not. The optional loop detection procedure is more fully and correctly specified here.

- Usage of tags is now mandatory (they were optional in RFC 2543), as they are now the fundamental building blocks of dialog identification.

- Added the Supported header field, allowing for clients to indicate what extensions are supported to a server, which can apply those extensions to the response, and indicate their usage with a Require in the response.

- Extension parameters were missing from the BNF for several header fields, and they have been added.

- Handling of Route and Record-Route construction was very underspecified in RFC 2543, and also not the right approach. It has been substantially reworked in this specification (and made vastly simpler), and this is arguably the largest change. Backwards compatibility is still provided for deployments that do not use "pre-loaded routes", where the initial request has a set of Route header field values obtained in some way outside of Record-Route. In those situations, the new mechanism is not interoperable.

- In RFC 2543, lines in a message could be terminated with CR, LF, or CRLF. This specification only allows CRLF.

- Usage of Route in CANCEL and ACK was not well defined in RFC 2543. It is now well specified; if a request had a Route header field, its CANCEL or ACK for a non-2xx response to the request need to carry the same Route header field values. ACKs for 2xx responses use the Route values learned from the Record-Route of the 2xx responses.

- RFC 2543 allowed multiple requests in a single UDP packet. This usage has been removed.

- Usage of absolute time in the Expires header field and parameter has been removed. It caused interoperability problems in elements that were not time synchronized, a common occurrence. Relative times are used instead.

- The branch parameter of the Via header field value is now mandatory for all elements to use. It now plays the role of a unique transaction identifier. This avoids the complex and bug-laden transaction identification rules from RFC 2543. A magic cookie is used in the parameter value to determine if the previous hop has made the parameter globally unique, and comparison falls back to the old rules when it is not present. Thus, interoperability is assured.

- In RFC 2543, closure of a TCP connection was made equivalent to a CANCEL. This was nearly impossible to implement (and wrong) for TCP connections between proxies. This has been eliminated, so that there is no coupling between TCP connection state and SIP processing.

- RFC 2543 was silent on whether a UA could initiate a new transaction to a peer while another was in progress. That is now specified here. It is allowed for non-INVITE requests, disallowed for INVITE.

- PGP was removed. It was not sufficiently specified, and not compatible with the more complete PGP MIME. It was replaced with S/MIME.

- Added the "sips" URI scheme for end-to-end TLS. This scheme is not backwards compatible with RFC 2543. Existing elements that receive a request with a SIPS URI scheme in the Request-URI will likely reject the request. This is actually a feature; it ensures that a call to a SIPS URI is only delivered if all path hops can be secured.

- Additional security features were added with TLS, and these are described in a much larger and complete security considerations section.

- In RFC 2543, a proxy was not required to forward provisional responses from 101 to 199 upstream. This was changed to MUST. This is important, since many subsequent features depend on delivery of all provisional responses from 101 to 199.

- Little was said about the 503 response code in RFC 2543. It has since found substantial use in indicating failure or overload conditions in proxies. This requires somewhat special treatment. Specifically, receipt of a 503 should trigger an attempt to contact the next element in the result of a DNS SRV lookup. Also, 503 response is only forwarded upstream by a proxy under certain conditions.

- RFC 2543 defined, but did no sufficiently specify, a mechanism for UA authentication of a server. That has been removed. Instead, the mutual authentication procedures of RFC 2617 are allowed.

- A UA cannot send a BYE for a call until it has received an ACK for the initial INVITE. This was allowed in RFC 2543 but leads to a potential race condition.

- A UA or proxy cannot send CANCEL for a transaction until it gets a provisional response for the request. This was allowed in RFC 2543 but leads to potential race conditions.

- The action parameter in registrations has been deprecated. It was insufficient for any useful services, and caused conflicts when application processing was applied in proxies.

- RFC 2543 had a number of special cases for multicast. For example, certain responses were suppressed, timers were adjusted, and so on. Multicast now plays a more limited role, and the protocol operation is unaffected by usage of multicast as opposed to unicast. The limitations as a result of that are documented.

- Basic authentication has been removed entirely and its usage forbidden.

- Proxies no longer forward a 6xx immediately on receiving it. Instead, they CANCEL pending branches immediately. This avoids a potential race condition that would result in a UAC getting a 6xx followed by a 2xx. In all cases except this race condition, the result will be the same - the 6xx is forwarded upstream.

- RFC 2543 did not address the problem of request merging. This occurs when a request forks at a proxy and later rejoins at an element. Handling of merging is done only at a UA, and procedures are defined for rejecting all but the first request.

### 28.2 Minor Functional Changes

- Added the Alert-Info, Error-Info, and Call-Info header fields for optional content presentation to users.

- Added the Content-Language, Content-Disposition and MIME-Version header fields.

- Added a "glare handling" mechanism to deal with the case where both parties send each other a re-INVITE simultaneously. It uses the new 491 (Request Pending) error code.

- Added the In-Reply-To and Reply-To header fields for supporting the return of missed calls or messages at a later time.

- Added TLS and SCTP as valid SIP transports.

- There were a variety of mechanisms described for handling failures at any time during a call; those are now generally unified. BYE is sent to terminate.

- RFC 2543 mandated retransmission of INVITE responses over TCP, but noted it was really only needed for 2xx. That was an artifact of insufficient protocol layering. With a more coherent transaction layer defined here, that is no longer needed. Only 2xx responses to INVITEs are retransmitted over TCP.

- Client and server transaction machines are now driven based on timeouts rather than retransmit counts. This allows the state machines to be properly specified for TCP and UDP.

- The Date header field is used in REGISTER responses to provide a simple means for auto-configuration of dates in user agents.

- Allowed a registrar to reject registrations with expirations that are too short in duration. Defined the 423 response code and the Min-Expires for this purpose.

## 29 Normative References

[1] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

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

[3] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[4] Rosenberg, J. and H. Schulzrinne, "SIP: Locating SIP Servers", RFC 3263, June 2002.

[5] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[6] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[8] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[9] Vaha-Sipila, A., "URLs for Telephone Calls", RFC 2806, April 2000.

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

[11] Freed, F. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[12] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[13] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with SDP", RFC 3264, June 2002.

[14] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[15] Postel, J., "DoD Standard Transmission Control Protocol", RFC 761, January 1980.

[16] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[17] 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.

[18] Troost, R., Dorner, S. and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

[19] 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.

[20] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[21] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[22] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

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

[24] Ramsdell B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[25] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[26] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

## 30 Informative References

[27] R. Pandya, "Emerging mobile and personal communication systems," IEEE Communications Magazine, Vol. 33, pp. 44--52, June 1995.

[28] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[29] Schulzrinne, H., Rao, R. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[30] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and J. Segers, "Megaco Protocol Version 1.0", RFC 3015, November 2000.

[31] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[32] Hoffman, P., Masinter, L. and J. Zawinski, "The mailto URL scheme", RFC 2368, July 1998.

[33] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California, Aug. 1996.

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

[35] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[36] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, September 1998.

[37] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000.

[38] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997.

[39] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP: Digest Access Authentication", RFC 2069, January 1997.

[40] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Willis, D., Rosenberg, J., Summers, K. and H. Schulzrinne, "SIP Call Flow Examples", Work in Progress.

[41] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing system," Journal of Internetworking: Research and Experience, Vol. 4, pp. 99--120, June 1993. ISI reprint series ISI/RS-93-359.

[42] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), (Berlin, Germany), Mar. 1996.

[43] Floyd, S., "Congestion Control Principles", RFC 2914, September 2000.

A Table of Timer Values

Table 4 summarizes the meaning and defaults of the various timers used by this specification.

| Timer | Value | Section | Meaning |
|---|---|---|---|
| T1 | 500ms default | Section 17.1.1.1 | RTT Estimate |
| T2 | 4s | Section 17.1.2.2 | The maximum retransmit interval for non-INVITE requests and INVITE responses |
| T4 | 5s | Section 17.1.2.2 | Maximum duration a message will remain in the network |
| Timer A | initially T1 | Section 17.1.1.2 | INVITE request retransmit interval, for UDP only |
| Timer B | 64*T1 | Section 17.1.1.2 | INVITE transaction timeout timer |
| Timer C | > 3min | Section 16.6 bullet 11 | proxy INVITE transaction timeout |
| Timer D | > 32s for UDP 0s for TCP/SCTP | Section 17.1.1.2 | Wait time for response retransmits |
| Timer E | initially T1 | Section 17.1.2.2 | non-INVITE request retransmit interval, UDP only |
| Timer F | 64*T1 | Section 17.1.2.2 | non-INVITE transaction timeout timer |
| Timer G | initially T1 | Section 17.2.1 | INVITE response retransmit interval |
| Timer H | 64*T1 | Section 17.2.1 | Wait time for ACK receipt |
| Timer I | T4 for UDP 0s for TCP/SCTP | Section 17.2.1 | Wait time for ACK retransmits |
| Timer J | 64*T1 for UDP 0s for TCP/SCTP | Section 17.2.2 | Wait time for non-INVITE request retransmits |
| Timer K | T4 for UDP 0s for TCP/SCTP | Section 17.1.2.2 | Wait time for response retransmits |

Table 4: Summary of timers

Acknowledgments

We wish to thank the members of the IETF MMUSIC and SIP WGs for their comments and suggestions. Detailed comments were provided by Ofir Arkin, Brian Bidulock, Jim Buller, Neil Deason, Dave Devanathan, Keith Drage, Bill Fenner, Cedric Fluckiger, Yaron Goland, John Hearty, Bernie Hoeneisen, Jo Hornsby, Phil Hoffer, Christian Huitema, Hisham Khartabil, Jean Jervis, Gadi Karmi, Peter Kjellerstedt, Anders Kristensen, Jonathan Lennox, Gethin Liddell, Allison Mankin, William Marshall, Rohan Mahy, Keith Moore, Vern Paxson, Bob Penfield, Moshe J. Sambol, Chip Sharp, Igor Slepchin, Eric Tremblay, and Rick Workman.

Brian Rosen provided the compiled BNF.

Jean Mahoney provided technical writing assistance.

This work is based, inter alia, on [41,42].

## Authors' Addresses

Authors addresses are listed alphabetically for the editors, the writers, and then the original authors of RFC 2543. All listed authors actively contributed large amounts of text to this document.

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Ave
East Hanover, NJ 07936
USA

EMail: jdrosen@dynamicsoft.com

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA

EMail: schulzrinne@cs.columbia.edu

Gonzalo Camarillo
Ericsson
Advanced Signalling Research Lab.
FIN-02420 Jorvas
Finland

EMail: Gonzalo.Camarillo@ericsson.com

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

EMail: alan.johnston@wcom.com

Jon Peterson
NeuStar, Inc
1800 Sutter Street, Suite 570
Concord, CA 94520
USA

EMail: jon.peterson@neustar.com

Robert Sparks
dynamicsoft, Inc.
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
USA

EMail: rsparks@dynamicsoft.com

Mark Handley
International Computer Science Institute
1947 Center St, Suite 600
Berkeley, CA 94704
USA

EMail: mjh@icir.org

Eve Schooler
AT&T Labs-Research
75 Willow Road
Menlo Park, CA 94025
USA

EMail: schooler@research.att.com

## Full Copyright Statement



## Acknowledgement

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