[MS-SIPAE]:

Session Initiation Protocol (SIP) Authentication Extensions

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1	New	Initial Availability
4/25/2008	0.2	Major	Revised and edited the technical content
6/27/2008	1.0	Major	Revised and edited the technical content
8/15/2008	1.01	Major	Revised and edited the technical content
12/12/2008	2.0	Major	Revised and edited the technical content
2/13/2009	2.01	Minor	Edited the technical content
3/13/2009	2.02	Minor	Edited the technical content
7/13/2009	2.03	Major	Revised and edited the technical content
8/28/2009	2.04	Editorial	Revised and edited the technical content
11/6/2009	2.05	Editorial	Revised and edited the technical content
2/19/2010	2.06	Editorial	Revised and edited the technical content
3/31/2010	2.07	Major	Updated and revised the technical content
4/30/2010	2.08	Editorial	Revised and edited the technical content
6/7/2010	2.09	Editorial	Revised and edited the technical content
6/29/2010	2.10	Editorial	Changed language and formatting in the technical content.
7/23/2010	2.10	None	No changes to the meaning, language, or formatting of the technical content.
9/27/2010	3.0	Major	Significantly changed the technical content.
11/15/2010	3.0	None	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	3.0	None	No changes to the meaning, language, or formatting of the technical content.
3/18/2011	3.0	None	No changes to the meaning, language, or formatting of the technical content.
6/10/2011	3.0	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	4.0	Major	Significantly changed the technical content.
4/11/2012	4.0	None	No changes to the meaning, language, or formatting of the technical content.
7/16/2012	4.0	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	4.1	Minor	Clarified the meaning of the technical content.
2/11/2013	4.1	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
7/30/2013	4.1	None	No changes to the meaning, language, or formatting of the technical content.
11/18/2013	4.1	None	No changes to the meaning, language, or formatting of the technical content.
2/10/2014	4.1	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	4.2	Minor	Clarified the meaning of the technical content.
7/31/2014	4.3	Minor	Clarified the meaning of the technical content.
10/30/2014	4.3	None	No changes to the meaning, language, or formatting of the technical content.
3/30/2015	5.0	Major	Significantly changed the technical content.
9/4/2015	5.0	None	No changes to the meaning, language, or formatting of the technical content.
7/15/2016	5.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Session Initiation Protocol (SIP) Authentication Extensions protocol. This protocol extends Session Initiation Protocol (SIP) for authentication functionality. SIP is used by terminals to establish, modify, and terminate multimedia sessions or calls.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

200 OK: A response to indicate that the request has succeeded.

403 Forbidden: A response that indicates that a protocol server understood but denies a request.

address-of-record: A **Session Initiation Protocol (SIP) URI** that specifies a domain with a location service that can map the URI to another URI for a user, as described in [RFC3261].

Augmented Backus-Naur Form (ABNF): A modified version of Backus-Naur Form (BNF), commonly used by Internet specifications. ABNF notation balances compactness and simplicity with reasonable representational power. ABNF differs from standard BNF in its definitions and uses of naming rules, repetition, alternatives, order-independence, and value ranges. For more information, see [RFC5234].

authentication: The act of proving an identity to a server while providing key material that binds the identity to subsequent communications.

base16: A binary-to-text encoding scheme whereby an arbitrary sequence of bytes is converted to a sequence of printable ASCII characters. Base16 uses only the digits 0 through 9 and the letters A through F.

base64 encoding: A binary-to-text encoding scheme whereby an arbitrary sequence of bytes is converted to a sequence of printable ASCII characters, as described in [RFC4648].

call: A communication between peers that is configured for a multimedia conversation.

certificate: A certificate is a collection of attributes (1) and extensions that can be stored persistently. The set of attributes in a certificate can vary depending on the intended usage of the certificate. A certificate securely binds a public key to the entity that holds the corresponding private key. A certificate is commonly used for **authentication** and secure exchange of information on open networks, such as the Internet, extranets, and intranets. Certificates are digitally signed by the issuing certification authority (CA) and can be issued for a user, a computer, or a service. The most widely accepted format for certificates is defined by the ITU-T X.509 version 3 international standards. For more information about attributes and extensions, see [RFC3280] and [X509] sections 7 and 8.

conference: A Real-Time Transport Protocol (RTP) session that includes more than one participant (2).

credential: Previously established, **authentication** data that is used by a security principal to establish its own identity. When used in reference to the Netlogon Protocol, it is the data that is stored in the NETLOGON_CREDENTIAL structure.

datagram: A style of communication offered by a network transport protocol where each message is contained within a single network packet. In this style, there is no requirement for establishing a session prior to communication, as opposed to a connection-oriented style.

delegate: A user or resource that has permissions to act on behalf of another user or resource.

delegator: A user or resource for which another user or resource has permission to act on its behalf.

dialog: A peer-to-peer **Session Initiation Protocol (SIP)** relationship that exists between two user agents and persists for a period of time. A dialog is established by **SIP messages**, such as a 2xx response to an INVITE request, and is identified by a call identifier, a local tag, and a remote tag.

digest: The fixed-length output string from a one-way hash function that takes a variable-length input string and is probabilistically unique for every different input string. Also, a cryptographic checksum of a data (octet) stream.

domain: A set of users and computers sharing a common namespace and management infrastructure. At least one computer member of the set must act as a **domain controller (DC)** and host a member list that identifies all members of the domain, as well as optionally hosting the Active Directory service. The domain controller provides **authentication** of members, creating a unit of trust for its members. Each domain has an identifier that is shared among its members. For more information, see [MS-AUTHSOD] section 1.1.1.5 and [MS-ADTS].

domain controller (DC): The service, running on a server, that implements Active Directory, or the server hosting this service. The service hosts the data store for objects and interoperates with other **DCs** to ensure that a local change to an object replicates correctly across all **DCs**. When Active Directory is operating as Active Directory Domain Services (AD DS), the **DC** contains full NC replicas of the configuration naming context (config NC), schema naming context (schema NC), and one of the domain NCs in its forest. If the AD DS **DC** is a global catalog server (GC server), it contains partial NC replicas of the remaining domain NCs in its forest. For more information, see [MS-AUTHSOD] section 1.1.1.5.2 and [MS-ADTS]. When Active Directory is operating as Active Directory Lightweight Directory Services (AD LDS), several AD LDS **DCs** can run on one server. When Active Directory is operating as AD DS, only one AD DS **DC** can run on one server. However, several AD LDS **DCs** can coexist with one AD DS **DC** on one server. The AD LDS **DC** contains full NC replicas of the config NC and the schema NC in its forest. The domain controller is the server side of Authentication Protocol Domain Support [MS-APDS].

endpoint: A device that is connected to a computer network.

focus: A single user agent that maintains a **dialog** and **Session Initiation Protocol (SIP)** signaling relationship with each participant (2), implements conference policies, and ensures that each participant receives the media that comprise the tightly coupled **conference**.

fully qualified domain name (FQDN): An unambiguous domain name (2) that gives an absolute location in the Domain Name System's (DNS) hierarchy tree, as defined in [RFC1035] section 3.1 and [RFC2181] section 11.

Generic Security Services (GSS): An Internet standard, as described in [RFC2743], for providing security services to applications. It consists of an application programming interface (GSS-API) set, as well as standards that describe the structure of the security data.

Globally Routable User Agent URI (GRUU): A **URI** that identifies a user agent and is globally routable. A URI possesses a GRUU property if it is useable by any **user agent client (UAC)** that is connected to the Internet, routable to a specific user agent instance, and long-lived.

hash: A fixed-size result that is obtained by applying a one-way mathematical function, which is sometimes referred to as a hash algorithm, to an arbitrary amount of data. If the input data changes, the hash also changes. The hash can be used in many operations, including **authentication** and digital signing.

Hash-based Message Authentication Code (HMAC): A mechanism for message **authentication** using cryptographic hash functions. HMAC can be used with any iterative cryptographic hash function (for example, **MD5** and **SHA-1**) in combination with a secret shared key. The cryptographic strength of HMAC depends on the properties of the underlying hash function.

INVITE: A **Session Initiation Protocol (SIP)** method that is used to invite a user or a service to participate in a session.

Kerberos: An **authentication** system that enables two parties to exchange private information across an otherwise open network by assigning a unique key (called a **ticket**) to each user that logs on to the network and then embedding these tickets into messages sent by the users. For more information, see [MS-KILE].

Key Distribution Center (KDC): The **Kerberos** service that implements the **authentication** and **ticket** granting services specified in the **Kerberos** protocol. The service runs on computers selected by the administrator of the realm or domain; it is not present on every machine on the network. It must have access to an account database for the realm that it serves. Windows **KDCs** are integrated into the **domain controller** role of a Windows Server operating system acting as a Domain Controller. It is a network service that supplies **tickets** to clients for use in authenticating to services.

master secret: A key that is used to symmetrically encrypt and decrypt credentials and single sign-on (SSO) tickets.

MD5: A one-way, 128-bit hashing scheme that was developed by RSA Data Security, Inc., as described in [RFC1321].

nonce: A number that is used only once. This is typically implemented as a random number large enough that the probability of number reuse is extremely small. A nonce is used in authentication protocols to prevent replay attacks. For more information, see [RFC2617].

NT LAN Manager (NTLM) Authentication Protocol: A protocol using a challenge-response mechanism for **authentication** in which clients are able to verify their identities without sending a password to the server. It consists of three messages, commonly referred to as Type 1 (negotiation), Type 2 (challenge) and Type 3 (authentication). For more information, see [MS-NLMP].

principal: (1) An identifier of such an entity.

(2) In **Kerberos**, a Kerberos principal.

proxy: A computer, or the software that runs on it, that acts as a barrier between a network and the Internet by presenting only a single network address to external sites. By acting as a go-between that represents all internal computers, the proxy helps protects network identities while also providing access to the Internet.

REGISTER: A **Session Initiation Protocol (SIP)** method that is used by an SIP client to register the client address with an SIP server.

security association (SA): A simplex "connection" that provides security services to the traffic carried by it. See [RFC4301] for more information.

Security Support Provider Interface (SSPI): A Windows-specific API implementation that provides the means for connected applications to call one of several security providers to establish authenticated connections and to exchange data securely over those connections. This is the Windows equivalent of Generic Security Services (GSS)-API, and the two families of APIs are on-the-wire compatible.

security token service (STS): A web service that issues claims (2) and packages them in encrypted security tokens.

server: A replicating machine that sends replicated files to a partner (client). The term "server" refers to the machine acting in response to requests from partners that want to receive replicated files.

Session Initiation Protocol (SIP): An application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. **SIP** is defined in [RFC3261].

SHA-1: An algorithm that generates a 160-bit hash value from an arbitrary amount of input data, as described in [RFC3174]. SHA-1 is used with the Digital Signature Algorithm (DSA) in the Digital Signature Standard (DSS), in addition to other algorithms and standards.

SHA-1 hash: A hashing algorithm as specified in [FIPS180-2] that was developed by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA).

SHA-256: An algorithm that generates a 256-bit hash value from an arbitrary amount of input data, as described in [FIPS180-2].

SIP element: An entity that understands the **Session Initiation Protocol (SIP)**.

SIP message: The data that is exchanged between **Session Initiation Protocol (SIP)** elements as part of the protocol. An SIP message is either a request or a response.

SIP protocol client: A network client that sends **Session Initiation Protocol (SIP)** requests and receives SIP responses. An SIP client does not necessarily interact directly with a human user. **User agent clients (UACs)** and proxies are SIP clients.

SIP registrar: A **Session Initiation Protocol (SIP)** server that accepts REGISTER requests and places the information that it receives from those requests into the location service for the domain that it handles.

SIP transaction: A **SIP transaction** occurs between a **UAC** and a **UAS**. The **SIP transaction** comprises all messages from the first request sent from the **UAC** to the **UAS** up to a final response (non-1xx) sent from the **UAS** to the **UAC**. If the request is **INVITE**, and the final response is a non-2xx, the **SIP transaction** also includes an ACK to the response. The ACK for a 2xx response to an **INVITE** request is a separate **SIP transaction**.

ticket: A record generated by the **key distribution center (KDC)** that helps a client authenticate to a service. It contains the client's identity, a unique cryptographic key for use with this ticket (the session key), a time stamp, and other information, all sealed using the service's secret key. It only serves to authenticate a client when presented along with a valid authenticator.

token: A word in an item or a search query that translates into a meaningful word or number in written text. A token is the smallest textual unit that can be matched in a search query. Examples include "cat", "AB14", or "42".

Transport Layer Security (TLS): A security protocol that supports confidentiality and integrity of messages in client and server applications communicating over open networks. **TLS** supports server and, optionally, client authentication by using X.509 certificates (as specified in [X509]). **TLS** is standardized in the IETF TLS working group. See [RFC4346].

tuple: An ordered grouping of members from different dimensions or hierarchies. A single member is a special case of a tuple and can be used as an expression. Every hierarchy does not have to be represented in a tuple.

Uniform Resource Identifier (URI): A string that identifies a resource. The URI is an addressing mechanism defined in Internet Engineering Task Force (IETF) Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

user agent client (UAC): A logical entity that creates a new request, and then uses the client transaction state machinery to send it. The role of **UAC** lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a **UAC** for the duration of that transaction. If it receives a request later, it assumes the role of a **user agent server (UAS)** for the processing of that transaction.

user agent server (UAS): A logical entity that generates a response to a **Session Initiation Protocol (SIP)** request. The response either accepts, rejects, or redirects the request. The role of the UAS lasts only for the duration of that transaction. If a process responds to a request, it acts as a UAS for that transaction. If it initiates a request later, it assumes the role of a **user agent client (UAC)** for that transaction.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[FIPS180-2] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-2, August 2002, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[MS-CONFBAS] Microsoft Corporation, "Centralized Conference Control Protocol: Basic Architecture and Signaling".

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions".

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol".

[MS-OCAUTHWS] Microsoft Corporation, "OC Authentication Web Service Protocol".

[MS-PRES] Microsoft Corporation, "Presence Protocol".

[MS-SIPRE] Microsoft Corporation, "Session Initiation Protocol (SIP) Routing Extensions".

[RFC2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, http://www.ietf.org/rfc/rfc2104.txt

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

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.rfc-editor.org/rfc/rfc2246.txt

[RFC2716] Aboba, B. and Simon, D., "PPP EAP TLS Authentication Protocol", RFC 2716, October 1999, http://www.ietf.org/rfc/rfc2716.txt

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000, http://www.rfc-editor.org/rfc/rfc2743.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.rfc-editor.org/rfc/rfc2818.txt

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E., "SIP: Session Initiation Protocol", RFC 3261, June 2002, http://www.ietf.org/rfc/rfc3261.txt

[RFC3323] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002, http://www.rfc-editor.org/rfc/rfc3323.txt

[RFC3325] Jennings, C., Peterson, J., and Watson, M., "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002, http://www.rfc-editor.org/rfc/rfc3325.txt

[RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003, http://www.rfc-editor.org/rfc/rfc3548.txt

[RFC3892] Sparks, R., "The Session Initiation Protocol (SIP) Referred-By Mechanism", RFC 3892, September 2004, http://www.rfc-editor.org/rfc/rfc3892.txt

[RFC4028] Donovan, S., and Rosenberg, J., "Session Timers in the Session Initiation Protocol (SIP)", RFC 4028, April 2005, http://www.rfc-editor.org/rfc/rfc4028.txt

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, http://www.rfc-editor.org/rfc/rfc4120.txt

[RFC4121] Zhu, L., Jaganathan, K., and Hartman, S., "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005, http://www.ietf.org/rfc/rfc4121.txt

[RFC5234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, http://www.rfc-editor.org/rfc/rfc5234.txt

1.2.2 Informative References

[RFC3324] Watson, M., "Short Term Requirements for Network Asserted Identity", RFC 3324, November 2002, http://www.rfc-editor.org/rfc/rfc3324.txt

1.3 Overview

This protocol specifies the **authentication** extensions to **Session Initiation Protocol (SIP)**. This protocol defines NT LAN Manager (NTLM), **Kerberos**, and Transport Layer Security with Derived Session Key (TLS-DSK) authentication schemes based on the general authentication framework described in [RFC3261], where the authentication mechanism is extended as described in Protocol Overview (section 3.1). This protocol also specifies the details and extensions for the Asserted Identity mechanism, which is based on [RFC3325], and the Referred-By mechanism, which is based on [RFC3892].

1.4 Relationship to Other Protocols

This protocol depends on SIP and makes use of the **NT LAN Manager (NTLM) Authentication Protocol**, as described in [MS-NLMP], and the Kerberos protocol, as described in [RFC4120] and [MS-KILE]. It also makes use of the **Transport Layer Security (TLS)** protocol, as described in [RFC2246].

1.5 Prerequisites/Preconditions

This protocol assumes that **SIP protocol clients** and the **server** support SIP. The prerequisites for this protocol are the same as the prerequisites for SIP.

1.6 Applicability Statement

This protocol is applicable when protocol clients and the server support SIP and intend to use one or more of the enhancements offered by this protocol.

1.7 Versioning and Capability Negotiation

Versions of this protocol prior to version 3 did not carry the version identifier in protocol messages. Versions of this protocol starting with version 3 carry a version identifier in the **authentication** header fields, as specified in section 2.2.1 and section 2.2.2. The differences between versions are covered in message processing sections, specifically 3.2.4.1, 3.2.5.1, 3.2.5.2, 3.2.5.3, 3.3.4.1, 3.3.5.2, and 3.3.5.3.

1.8 Vendor-Extensible Fields

There are no vendor-extensible fields specific to this protocol. Standard extension mechanisms of the SIP can be used by vendors as needed.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The messages specified by this protocol are carried in the SIP authentication exchanges in SIP **authentication** headers. This protocol does not introduce a new transport to exchange messages and is capable of being used with any transport used by SIP.

2.2 Message Syntax

This protocol relies on the **SIP message** format, as specified in [RFC3261] section 25.1, and extends definitions of authentication-related header fields, specifically **Authentication-Info**, **Authorization**, **Proxy-Authenticate**, **Proxy-Authorization**, and **WWW-Authenticate**, as well as the **Referred-By** header field, which is specified in [RFC3892]. This protocol defines two new header fields, **Proxy-Authentication-Info** and **p-session-on-behalf-of**, and makes use of **endpoint** identification extensions that are specified in [MS-SIPRE].

All of the message syntax specified in this protocol is described in both prose and an **Augmented Backus-Naur Form (ABNF)** defined in [RFC5234].

2.2.1 WWW-Authenticate and Proxy-Authenticate Response Header Fields

[RFC3261] Section 25.1 defines the syntax for the **WWW-Authenticate** and **Proxy-Authenticate** header fields as follows.

```
Proxy-Authenticate  = "Proxy-Authenticate" HCOLON challenge
WWW-Authenticate    = "WWW-Authenticate" HCOLON challenge

challenge           = ("Digest" LWS digest-cln *(COMMA digest-cln))
                      / other-challenge
```

This protocol defines the following extensions.

```
challenge           = ("Digest" LWS digest-cln *(COMMA digest-cln))
                      / "NTLM" LWS msspi-cln *(COMMA msspi-cln)
                      / "Kerberos" LWS msspi-cln *(COMMA msspi-cln)
                      / "TLS-DSK" LWS msspi-cln *(COMMA msspi-cln)
                      / other-challenge
digest-cln          = realm / domain / nonce
                      / opaque / stale / algorithm
                      / qop-options / auth-param
algorithm           = "algorithm" EQUAL
                      ( "MD5" / "MD5-sess"/ "SHA256-sess" / token )
msspi-cln           = realm / opaque
                      / targetname / gssapi-data / version / sts-uri
targetname          = "targetname" EQUAL target-value
target-value        = DQUOTE ( ntlm-target-val
                      / ( "sip/" kerberos-target-val)
                      / tls-dsk-target-val ) DQUOTE
ntlm-target-val     = token
kerberos-target-val = token
tls-dsk-target-val  = token
gssapi-data         = "gssapi-data" EQUAL gssapi-data-value
gssapi-data-value   = quoted-string
version             = "version" EQUAL version-value
version-value       = 1*DIGIT
sts-uri             = "sts-uri" EQUAL DQUOTE absoluteURI DQUOTE
```

The **other-challenge**, **LWS**, **COMMA**, **realm**, **domain**, **nonce**, **opaque**, **stale**, **qop-options**, **auth-param**, **EQUAL**, **quoted-string**, **token**, **absoluteURI**, and **DIGIT** attributes are defined in [RFC3261] Section 25.

The **ntlm-target-val**, **kerberos-target-val**, and **tls-dsk-target-val** values carry a **token** that uniquely identifies the SIP server among all the possible **principals (1)** within the NTLM, Kerberos, and TLS-DSK authentication protocol namespaces.

The **gssapi-data-val** value carries the cryptographic token generated by the authentication protocol implementation.

The **version-value** value carries the version number of the protocol.

2.2.2 Authentication-Info and Proxy-Authentication-Info Header Fields

[RFC3261] Section 25 defines the syntax for the **Authentication-Info** header field.

```
Authentication-Info = "Authentication-Info" HCOLON ainfo
                      *(COMMA ainfo)
ainfo               =  nextnonce / message-qop
                       / response-auth / cnonce
                       / nonce-count
```

This protocol defines a new header field, **Proxy-Authentication-Info**, and the following extensions for the **Authentication-Info** and **Proxy-Authentication-Info** header fields.

```
Authentication-Info  = "Authentication-Info" HCOLON
                       ("NTLM" / "Kerberos" / "TLS-DSK")
                       LWS ainfo *(COMMA ainfo)
Proxy-Authentication-Info = "Proxy-Authentication-Info" HCOLON
                       ("NTLM" / "Kerberos" / "TLS-DSK")
                       LWS ainfo *(COMMA ainfo)

ainfo               = nextnonce / message-qop
                       / response-auth / cnonce
                       / nonce-count
                       / snum / srand
                       / realm / targetname / opaque / version

snum                = "snum" EQUAL snum-value
snum-value          = 1*DIGIT / DQUOTE 1*DIGIT DQUOTE
srand               = "srand" EQUAL srand-value
srand-value         = 8LHEX / DQUOTE 8HEXDIG DQUOTE
version             = "version" EQUAL version-value
version-value       = 1*DIGIT
```

The **nextnonce**, **message-qop**, **response-auth**, **cnonce**, **nonce-count**, **realm**, **opaque**, **EQUAL**, **DIGIT**, **DQUOTE**, and **HEXDIG** attributes are defined in [RFC3261] Section 25.

The **targetname** attribute is defined in section 2.2.1.

The **version-value** value carries the version number of the protocol.

[RFC3261] section 20 specifies that the **Authentication-Info** header field MUST NOT be used in any but the **200 OK** response, and MUST NOT be used in ACK and CANCEL requests. This protocol allows the **Authentication-Info** header field in any response and in ACK and CANCEL requests. It also allows **Proxy-Authentication-Info** in any request or response.

2.2.3 Authorization and Proxy-Authorization Header Fields

[RFC3261] section 25.1 defines the syntax for the **Authorization** and **Proxy-Authorization** header fields as follows.

```
Authorization       = "Authorization" HCOLON credentials
Proxy-Authorization = "Proxy-Authorization" HCOLON credentials
credentials         = ("Digest" LWS digest-response)
                       / other-response
```

This specification defines the following extensions:

```
credentials         = ("Digest" LWS digest-response)
                       / ("NTLM" LWS msspi-response)
                       / ("Kerberos" LWS msspi-response)
                       / ("TLS-DSK" LWS msspi-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
algorithm           = "algorithm" EQUAL ( "MD5" / "MD5-sess"
                      / "SHA256-sess" / token )
msspi-response      = msspi-resp *(COMMA msspi-resp)
msspi-resp          = message-qop / realm / opaque
                      / version / targetname / gssapi-data
                      / crand / cnum / msspi-resp-data
cnum                = "cnum" EQUAL cnum-value
cnum-value          = 1*DIGIT / DQUOTE 1*DIGIT DQUOTE
crand               = "crand" EQUAL crand-val
crand-val           = 8LHEX / DQUOTE 8LHEX DQUOTE
msspi-resp-data     = "response" EQUAL msspi-resp-data-value
msspi-resp-data-val = quoted-string
```

The **LWS**, **COMMA**, **realm**, **opaque**, **EQUAL**, **quoted-string**, **DIGIT**, **DQUOTE**, **LHEX**, **username**, **nonce**, **digest-uri**, **dresponse**, **cnonce**, **message-qop**, **nonce-count**, **auth-param**, and **other-response** attributes are defined in [RFC3261] section 25.1.

The **targetname** and **version** attributes are defined in section 2.2.1.

The **gssapi-data** attribute is defined in section 2.2.1.

[RFC3261] section 20 specifies that the **Authorization** and **Proxy-Authorization** header fields MUST NOT be used in responses. A **Proxy-Authorization** header field MUST NOT be used in a CANCEL request. This protocol allows these header fields in any response, as well as in a CANCEL request.

2.2.4 Endpoint Identification Extensions

This specification makes use of the following endpoint identification extensions defined in [MS-SIPRE].

- **epid** parameter in **From** and **To** header fields,
- **+sip.instance** parameter in a **Contact** header field,
- **Globally Routable User Agent URI (GRUU)** as the **Uniform Resource Identifier (URI)** of the **Contact** header field.

2.2.5 Referred-By Header Field Extensions

This protocol extends the syntax of the **ReferredBy** header field defined in [RFC3892] section 3. The extensions to the original ABNF, as defined in [RFC5234], are as follows.

```
Referred-By  =  ("Referred-By" / "b") HCOLON referrer-uri
                  *( SEMI (referredby-id-param
                         / ms-identity-param
                         / ms-identity-alg-param
                         / ms-identity-info-param
                         / ms-identity-cookie-param
                         / ms-referee-uri-param
                         / generic-param) )
ms-identity-param = "ms-identity" EQUAL ms-identity-token
ms-identity-token = quoted-string
ms-identity-alg-param = "ms-identity-alg" EQUAL token
ms-identity-info-param = "ms-identity-info" EQUAL ms-identity-info-val
ms-identity-info-val = quoted-string
ms-identity-cookie-param = "ms-identity-cookie" EQUAL
                           ms-identity-cookie-token
ms-identity-cookie-token = quoted-string
ms-referee-uri-param = "ms-referee-uri" EQUAL DQUOTE SIP-URI DQUOTE
```

EQUAL, **generic-param**, **quoted-string**, and **token** are defined in [RFC3261] section 25.1.

Referrer-uri and **referredby-id-param** are defined in [RFC3892] section 3**.**

The **ms-referee-uri-param** parameter used in the protocol between the SIP protocol client and the server is documented in section 3.2.4.4. Other extension parameters are used only for communications between SIP servers and have the following definitions.

- **ms-identity-param:** A token that cryptographically verifies the identity of the referrer within the context of the referred **call**.
- **ms-identity-alg-param:** A token that specifies the cryptographic algorithm used for the **ms-identity-param** computation.
- **ms-identity-info-param:** Information about the server that performed the **ms-identity-param** computation.
- **ms-identity-cookie-param:** Cryptographic token that verifies the content of the **ReferredBy** header field.

2.2.6 p-session-on-behalf-of Header Field Syntax

This protocol defines a new header field called **p-session-on-behalf-of**. This header field is populated in a message by a client when it wishes to convey to the target of the message that it is acting on behalf of another user. The value of the header field is the **address-of-record** of the user, or **delegator**, that it is acting on behalf of, and the client initiating this message is the **delegate**. Setting up the **delegate** relation is specified in [MS-PRES]. The ABNF for it, as defined in [RFC5234], is:

```
p-session-on-behalf-of = "p-session-on-behalf-of" HCOLON
                         ( name-addr / addr-spec )
```

The **name-addr** and **addr-spec** are defined in [RFC3261] section 25.

3 Protocol Details

3.1 Protocol Overview

This protocol implements a proprietary Kerberos, NTLM, and TLS-DSK<1> authentication mechanism that is used by the SIP protocol client for client-to-server authentication and mutual signing of messages by both the SIP client and the SIP server. For more information about NTLM, see [MS-NLMP]. For more information about Kerberos, see [RFC4120] and [MS-KILE]. For more information about TLS, see [RFC2246].

Authentication consists of two phases. In the first phase, a **security association (SA)** is established between the protocol client and the server. In the second phase, the protocol client and server use the existing SA to sign messages that they send, and to verify the messages that they receive. The exact message exchange in the first phase differs depending on whether NTLM, Kerberos, or TLS-DSK authentication is used.

The primary distinction between NTLM and Kerberos is the need for connectivity to the **domain controller (DC)**. In Kerberos, the protocol client MUST request a Kerberos **ticket** from the **Key Distribution Center (KDC),** which in the proprietary implementation is a process that resides on the DC. In NTLM, the server verifies the protocol client's **credentials** by contacting the DC. This difference allows protocol clients that do not have connectivity to the DC to authenticate with the server using NTLM authentication, and it is the main reason for supporting NTLM in addition to the more secure and standard Kerberos authentication.

The TLS-DSK authentication is based on **certificates** that SHOULD be obtained by the client through an out-of-band mechanism, such as contacting a **security token service (STS)**. One such mechanism is specified in [MS-OCAUTHWS].

During the NTLM SA establishment phase, a three-way handshake, or three round trips, occurs between the protocol client and the protocol server.

- The protocol client sends a request with no credential or authentication (2) information. The server (2) responds to that request with a 401 Unauthorized or 407 Proxy Authentication Required, indicating that it supports NTLM and possibly other protocols, such as Kerberos or TLS-DSK, and requires authentication (2).
- The protocol client reissues the request, indicating its preference for NTLM authentication and including the content of **NTLM NEGOTIATE_MESSAGE**, as described in [MS-NLMP]**.** The server responds with an **NTLM CHALLENGE_MESSAGE** in a 401 Unauthorized or 407 Proxy Authentication Required.
- The protocol client reissues the request with an NTLM response, an **AUTHENTICATE_MESSAGE**, to the server's challenge. If the protocol client negotiates version 4 of the authentication protocol, it MUST also sign the request, or include **NTLMSSP_MESSAGE_SIGNATURE**. The server processes the request and responds, including **NTLMSSP_MESSAGE_SIGNATURE** for the response.
- The SA is now established on both the protocol client and server, and subsequent messages between the protocol client and server are signed, which means that they carry a signature formatted as **NTLMSSP_MESSAGE_SIGNATURE** in the message.

During the Kerberos SA establishment phase, a two-way handshake, or two round trips, occurs between the SIP protocol client and the SIP server (2).

- The protocol client sends a request with no credential or authentication (2) information. The server (2) responds to that request with a 401 Unauthorized or 407 Proxy Authentication Required, indicating that it supports Kerberos and possibly other protocols, such as NTLM or TLS-DSK, and requires authentication (2).

- The protocol client requests a Kerberos ticket for the server from the KDC, and reissues the request with this encoded Kerberos ticket information, or **KRB_AP_REQ**, as defined in [RFC4120]. If the protocol client negotiates version 4 of the authentication (2) protocol, it MUST also sign the request, or include a Kerberos signature.
- The server (2) processes the request and responds, including a Kerberos signature for the response.
- The SA is now established on both the protocol client and server, and subsequent messages between the protocol client and server are signed, which means that they carry an MIC token, as defined in [RFC4121], in the message.

During the TLS-DSK SA establishment phase, a four-way handshake, or four round trips, occurs between the SIP protocol client and the SIP server (2).

- The protocol client sends a request with no credential or authentication information. The server responds to that request with a 401 Unauthorized or 407 Proxy Authentication Required, indicating that it supports TLS-DSK, and possibly other protocols, such as NTLM or Kerberos, and requires authentication. The response from the server SHOULD include a URI of a STS that the client can contact to obtain a certificate for authentication using the TLS-DSK protocol.
- The protocol client locates or obtains a certificate and reissues the request, indicating its preference for TLS-DSK authentication and including data that encapsulates one or more TLS records in TLS record layer format, typically containing a TLS **client_hello** handshake message, as specified in [RFC2246]**.** The server responds with a 401 Unauthorized or 407 Proxy Authentication Required response that encapsulates one or more TLS records that contain a TLS **server_hello** handshake message, followed by a TLS certificate, such as **server_key_exchange**, and then **certificate_request**, **server_hello_done** handshake messages.
- The protocol client processes the response, reissues the request with data that encapsulates one or more TLS records containing TLS **certificate**, **client_key_exchange**, **certificate_verify**, **change_cipher_spec**, and **finished** handshake messages. The server processes the request, verifies the client certificate, and responds with a 401 Unauthorized or 407 Proxy Authentication Required response that encapsulates one or more TLS records that contain TLS **change_cipher_spec** and **finished** handshake messages.
- The protocol client processes the response, and computes, or derives, client and server signing keys from the TLS-negotiated key material using an algorithm similar to the one specified in [RFC2716], and reissues the request with the signature. The server (2) also computes, or derives, the server (2) and client signing keys using the same algorithm and verifies the signature in the request. It can now respond and include the signature.
- The SA is now established on both the protocol client and server (2), and subsequent messages between the protocol client and server (2) are signed, which means that they carry a signature computed with the client and server (2) signing keys.

For each SA, both the SIP protocol client and the server (2) MUST keep track of the message sequence numbers by maintaining a sliding window. The initial range of this window is 1 to 256, and it is adjusted upward based on the highest sequence number received, while maintaining a window size of 256. Messages within the window can arrive in any order with regard to their sequence number, as long as no sequence number is used more than once. The purpose of maintaining this sliding window is to provide replay protection while allowing pipelining of messages for performance reasons.

3.1.1 Abstract Data Model

None.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

None.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 SIP Client Details

3.2.1 Abstract Data Model

The protocol client establishes and maintains an SA with each protocol server that challenges it. The security association data includes the following attributes.

Authenticating server identification: A tuple that consists of a **realm** parameter value and authentication target derived from the **targetname** parameter value from the **authentication** header field **WWW-Authenticate or Proxy-Authenticate** of the challenge received by the protocol client when the SA was created. For the NTLM and TLS-DSK<2> authentication protocols, the authentication target is the value of the **targetname** parameter without quotes. For the Kerberos authentication protocol, the authentication target is the value of **targetname** without quotes and the **sip/** service descriptor.

Authenticating server protocol version: The value of the **version** parameter from the **authentication** header field of the challenge received by the protocol client when the SA was created.

Authenticating server opaque value: The value of the **opaque** parameter from the **authentication** header of the challenge received by the protocol client when the SA was created or during the "establishing" state.

Whether the authenticating server is a proxy or a user agent server (UAS): If the authenticating server used the **Proxy-Authenticate** header field in its challenge, it is a proxy. If it used **WWW-Authenticate**, it is a UAS.

State or phase: The "establishing" state occurs during an NTLM, TLS-DSK, or Kerberos authentication handshake. The "established" state occurs after NTLM, Kerberos, or TLS-DSK authentication completes and the protocol client signs outgoing messages and verifies signatures on incoming messages. The "expired" state occurs when the expiration timer fires.

SA expiration time: The time when the SA expires and needs to be recreated.

Outgoing message sequence number counter: The counter that is incremented every time the message is sent with an **authorization** header created using this SA.

High sequence number (SnumHigh): The highest sequence number for messages received from the server so far and verified using this SA.

The list of received sequence numbers within the sliding window: All sequence numbers extracted from messages sent by the server that fit into the sliding window between **SnumHigh-256** and **SnumHigh**.

Authentication protocol context: The context information used by an authentication protocol that is compliant with the **Generic Security Services (GSS)** Application Programming Interface, as specified in [RFC2743], and further clarified for NTLM in [MS-NLMP] and for Kerberos in [MS-KILE]. The context allows the authentication protocol to perform an authentication handshake when the SA is still in the "establishing" state and to sign outgoing messages or verify the integrity of incoming messages using an attached signature when the SA enters the "established" state. For the TLS-DSK authentication protocol, when the SA is still in the "establishing" state, the client maintains enough context information to process TLS messages, as specified in [RFC2246]. Once the SA enters the "established" state, the client context maintains the **hash** function from the ciphersuite negotiated by TLS, and client and server authentication keys generated at the end of the TLS negotiation.

The protocol client maintains a table of SAs it has established, indexed by authenticating server (2) identity.

The protocol client links each SA with one or more proxies or servers (2) from which it received the challenge, with the authentication (2) server (2) identity matching the one stored in the SA.

3.2.2 Timers

When the NTLM, Kerberos, or TLS-DSK<3> authentication handshake completes and the SA enters the "established" state, the SIP protocol client MUST start an SA expiration timer. For an SA established using NTLM, the expiration timer value is eight (8) hours reduced by some buffer time. For an SA established using the Kerberos authentication protocol, the protocol client MUST also retrieve the service ticket expiry time, as specified in [MS-KILE], when the SA enters the "established" state. The expiration timer value is the lesser of the service ticket expiry time and eight hours, further reduced by some buffer time. For an SA established using the TLS-DSK authentication protocol, the client MUST retrieve the expiration time of its certificate. The expiration timer value is the lesser of the interval to the certificate (1) expiration and eight hours, further reduced by some buffer time.

The protocol client MUST choose a sufficient buffer time to allow for the NTLM, Kerberos, or TLS-DSK authentication handshake that reestablishes the SA to complete before the eight-hour SA expiration time that is maintained by the SIP server, and before the Kerberos ticket expiry time or certificate (1) expiration time. This value SHOULD be five (5) minutes or longer.

3.2.3 Initialization

If the SIP protocol client does not already have a security association linked with the SIP server to which it intends to send the message for the first time, the protocol client SHOULD initiate the authentication handshake by sending a **REGISTER** request without any authorization, **Authorization** or **Proxy-Authorization**, header fields. The protocol client can use other requests, such as **INVITE** or SUBSCRIBE, also without **authorization** header fields. The protocol client SHOULD NOT use ACK or CANCEL requests or other requests that the SIP protocol does not allow to challenge. The request that initiates the authentication handshake MUST include a protocol client endpoint identifier, such as "epid", "+sip.instance", or "GRUU", as specified in [MS-SIPRE].

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Sending Messages to the SIP Server

When a SIP protocol client needs to send a message to a server or proxy, it MUST check for SAs that were established as the result of challenges received from this server or proxy, such as SAs linked to the server. If there are SAs in the "establishing" state linked to the same server, the protocol client SHOULD postpone sending the message until authentication handshakes for such SAs are complete. If all SAs linked to the same server are in the "established" state, the protocol client MUST use each of the SAs to generate and insert an **authorization** header field using the following steps.

1. The protocol client increments its outgoing message sequence number counter and generates a protocol client random value.

   The protocol client outgoing sequence number, **cnum**, is maintained on a per-SA basis and incremented each time this procedure is performed. It is stored as an unsigned decimal number, as described in ABNF in section 2.2.3.

   A protocol client random value, **crand**, is a 32-bit **nonce**, which is a random number large enough that the probability of number re-use is vanishingly small. It is stored as an eight-digit hexadecimal number, as described in ABNF in section 2.2.3.

2. The protocol client constructs a buffer with the information from the message and the SA that will be used in signature computation.

   The buffer is constructed from the following string values encoded in UTF8, in order, each enclosed by angle brackets (<>) with the same syntax and case (even if the field is case-insensitive) as when they appear in the message header fields:

   1. Authentication (2) protocol ("NTLM", "Kerberos", or "TLS-DSK").
   2. **crand** value as an eight-digit hexadecimal number.
   3. **cnum** value as a decimal number.
   4. **realm** parameter value without quotes as it appears in the challenge message sent by the server when the SA was created.
   5. **targetname** parameter value without quotes as it appears in the challenge message sent by the server when the SA was created.
   6. The value of the **Call-ID** header field from the message.
   7. The sequence number from the **CSeq** header field.
   8. The method from the **CSeq** header field.
   9. The URI in the **From** header field.
   10. The **tag** parameter value from the **From** header field.
   11. If the authenticating server protocol version is 3 or higher, the URI in the **To** header field.
   12. The **tag** parameter value from the **To** header field.
   13. If the authenticating server protocol version is 3 or higher, the "sip" URI from the **P-Asserted-Identity** or **P-Preferred-Identity** header field.
   14. If the authenticating server protocol version is 3 or higher, the "tel" URI from the **P-Asserted-Identity** or **P-Preferred-Identity** header field.

15. The value of the **Expires** header field.

16. If the message is a response, the response code value as a decimal string.

If a field mentioned in the list does not exist in the message, an empty string enclosed by angle brackets is included. This does not apply to fields included conditionally depending on protocol version or message type; an empty string enclosed by angle brackets is not needed if the condition is not satisfied. However, empty angle brackets are included if the condition is satisfied, but there is no corresponding header field in the message.

3. The protocol client uses an authentication protocol **GSS_GetMIC()** call, as specified in [MS-NLMP] section 3.1.4 for NTLM, and in [RFC2743] section 2.3.1 for Kerberos, to generate a signature token for the buffer constructed in the preceding step 2 using the authentication protocol context stored in the SA.

   Note that for the NTLM **Security Support Provider Interface (SSPI)**, the protocol client provides a fixed message sequence number of 100 in addition to the buffer and protocol context.

   For TLS-DSK<4>, the client computes the signature token using the **Hash-based Message Authentication Code (HMAC)** algorithm specified in [RFC2104], with the hash function and client authentication key obtained when the TLS negotiation completed, which means that the **finished** handshake message was received from the server, as described in section 3.3.5.1, and the buffer constructed in the preceding step 2.

4. The binary token returned by the authentication protocol implementation in the preceding step 3 is then encoded using the Base16 encoding procedure specified in [RFC3548] section 6. The characters 'A' through 'F' in the output of the Base16 encoding procedure SHOULD be replaced with their lowercase equivalents ('a' through 'f').

5. The protocol client generates an **Authorization** header field if the challenge that established the SA contained a **WWW-Authenticate** header field or a **Proxy-Authorization** header field if the challenge that established the SA contained a **Proxy-Authenticate** header field, according to the syntax described in section 2.2.3, with the data generated in the preceding steps and the data copied from the challenge message sent by the server (2) when the SA was established.

   Specifically, it adds the following fields:

   1. Authentication protocol ("NTLM", "Kerberos", or "TLS-DSK").

   2. **realm** with the value copied from the challenge message sent by the server when the SA was created.

   3. **targetname** with the value copied from the challenge message sent by the server when the SA was created.

   4. **opaque** with the value copied from the challenge message sent by the server when the SA was created.

   5. **qop** with the value "auth".

   6. **cnum** with the value generated in step 1.

   7. **crand** with the value generated in step 1.

   8. **response** with the value generated in step 4.

3.2.4.2 Communicating Alternate Identities in the Messages Sent to the SIP Server

The SIP protocol client might need to communicate a user's identity in addition to one in the address-of-record of the **From** header field of the request, or the **To** header field of the response. Moreover, if

the request is forwarded, or re-targeted, from one user to another, the address-of-record in the **To** header field might represent the user who was called originally and not the user who responds to the request, so the protocol client that formulates the response might need to insert the correct identity of the responding user without modifying the URI in the **To** header field.

In both of the preceding cases, the protocol client SHOULD add a **P-Preferred-Identity** header field with one or two values. If there is one value, it MUST be a **sip** or **tel** URI. If there are two values, one of them MUST be a **sip** URI and the other MUST be a **tel** URI.

If the identity inserted in the **P-Preferred-Identity** header field by a protocol client is not to be communicated outside the trust **domain** with which it authenticated, the client SHOULD insert a **Privacy** header field with the value "id", as described in [RFC3325] section 9.3. If, however, the client identity in the **P-Preferred-Identity** header field is to propagate outside the trust domain, the client SHOULD insert a **Privacy** header field with the value "none", as described in [RFC3323]. The definition of trust domain is based on [RFC3324] and comprises the network of securely interconnected server nodes.

3.2.4.3 Establishing session as anonymous client

The client MUST<5> only establish anonymous sessions for joining **conferences** of the type specified in [MS-CONFBAS].

When a client is required to establish a session with a remote entity that has a URI in the form of a conference GRUU, as specified in [MS-SIPRE], and the client does not have a registered endpoint, the client MUST establish the aforementioned session by creating a random anonymous URI and use that as the **From:** header of the sessions established. The client MUST use a URI of the form <username>@anonymous.invalid.

3.2.4.4 Specifying Referee Identity in the Referred-By Header Field in Forwarded/Retargeted Calls

When a **dialog-establishing** request is forwarded, or retargeted, from one user or phone number to another, the address-of-record in the URI of the **To** header field in **mid-dialog** requests might still reflect the SIP identity of the original user or phone number, before forwarding or retargeting. If such a **mid-dialog** request, with the original address-of-record or phone number in the URI of the **To** header field, is a REFER request and the protocol client sending it intends to communicate to the SIP server the address-of-record that represents the identity of the actual message recipient after forwarding or retargeting, it SHOULD insert the **ms-referee-uri** parameter with the value of the SIP URI representing the address-of-record of the actual message recipient into the **Referred-By** header field in the REFER request. See section 2.2.5 for the specifications of the **ms-referee-uri** parameter.

3.2.4.5 Specifying p-session-on-behalf-of Header

This section follows the product behavior described in product behavior note <6>.

If a user has been configured as a delegate of another user, which is referred to here as the delegator, and the **UAC** of the delegate sends an INVITE on behalf of the delegator, it SHOULD set the **p-session-on-behalf-of header** field in the INVITE. The value of the **p-session-on-behalf-of** header field SHOULD be set to the address-of-record of the delegator user agent.

When a request intended for a delegator is routed to a delegate client endpoint and the delegate client accepts the request on behalf of the delegator, it SHOULD set the **p-session-on-behalf-of** header field in the 2xx response to indicate that the request has been responded to on the delegator's behalf. The value of the **p-session-on-behalf-of** header field SHOULD be set to the address-of-record of the delegator.

For more information about setting up delegates, see [MS-PRES].

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Processing Challenges from the SIP Server

When a SIP protocol client receives a 401 Unauthorized or 407 Proxy Authentication Required response, or challenge, to the request that it previously sent to the server, it MUST examine **authentication** headers **WWW-Authenticate** and **Proxy-Authenticate** in the response. If authentication headers are for Digest authentication protocol, the challenge SHOULD<7> be handled as described in the section 3.2.5.5 and the rest of this section SHOULD NOT be used. Otherwise, the client MUST attempt to locate an existing security association that was created from the challenge that had the same **realm** parameter value and the same authentication target. For the NTLM and TLS-DSK<8> authentication protocols, the authentication target is the value of the **targetname** parameter without quotes. For the Kerberos authentication protocol, the authentication target is the **targetname** value without quotes and the **sip/** service descriptor. The protocol client MUST then process the response, or challenge, as follows.

1. The protocol client examines the **Date** header field in the challenge. The difference between the value in the **Date** header field inserted by the server and the current date and time at the protocol client could result in the server's failure to authenticate the protocol client. The protocol client prompts the user to synchronize the date with the server if there is a difference between the **Date** header field value and the current date and time at the protocol client, referred to as the clock skew, in excess of the maximum allowed by the authentication protocol, and the server fails to validate the user's credentials. As specified in [MS-KILE], the default acceptable clock skew for Kerberos is 5 minutes. No maximum time difference value is defined for NTLM and TLS-DSK.

2. If the protocol client finds the SA with a matching **realm** and authentication target, and the SA is in the "established" state, the protocol client determines whether the SA was used to sign the request to which the server responded with the challenge. The signing procedure is described in section 3.2.4.1.

   If the request is not signed, the protocol client resends the request with the signature and stops processing this challenge.

   If the request is already signed, the protocol client destroys the SA and proceeds to step 4.

   If the protocol client finds the SA with a matching **realm** and authentication target and the SA is not yet established, the protocol client determines whether the **authentication** header field is for the NTLM or TLS-DSK protocol and whether it contains a **gssapidata** parameter.

3. If the **authentication** header field is for the NTLM protocol and it contains a **gssapi-data** parameter, the protocol client decodes its value using the **base64** decoding procedure, as specified in [RFC3548] section 3, and passes it, along with the authentication protocol context associated with the SA and the value of the **targetname** parameter in the **GSS_Init_sec_context** call of the NTLM implementation, as specified in [MS-NLMP] section 3.1.4. This primitive in the NTLM protocol implementation generates the **AUTHENTICATE_MESSAGE**, and the protocol client then proceeds to step 5 to encode it and send it to the server (2).

   If the **authentication** header field is for the TLS-DSK protocol and it contains the **gssapi-data** parameter, the client decodes its value using the base64 decoding procedure, as specified in [RFC3548] section 3, and passes it, along with the current TLS context, to the TLS protocol implementation for processing. If the **gssapi-data** parameter value carried the TLS **server_hello** handshake message, followed by a TLS certificate, and possibly **server_key_exchange**, and then **certificate_request**, and **server_hello_done** handshake messages, the client validates the server certificate, obtains or locates a previously obtained client certificate, and generates an output token that carries TLS **certificate**, **client_key_exchange**, **certificate_verify**, **change_cipher_spec**, and **finished** handshake messages, as specified in [RFC2246]. If the **gssapi-data** parameter value carried TLS **change_cipher_spec** and **finished** handshake

messages, the client verifies that authentication and key exchanges were successful, and notes the cryptographic hash function selected by TLS ciphersuite negotiation, such as **MD5**, **SHA-1**, or **SHA-256**. The client then generates client and server authentication (2) keys as follows.

1. Given the **master secret** negotiated by the TLS handshake, the pseudo-random function (PRF) defined in the specification for the version of TLS in use, and the value randomly defined as the concatenation of the handshake message fields **client_hello.random** and **server_hello.random**, in that order, the value PRF (master secret, "client EAP encryption", random) is computed up to 128 bytes.

2. The client authentication (2) key used for computing and validating signatures for messages from client to server (2) is obtained by truncating to the correct length the third 32 bytes of the PRF output string.

3. The server (2) authentication (2) key used for computing and validating signatures for messages from server (2) to client is obtained by truncating to the correct length the fourth 32 bytes of this same PRF output string.

The preceding key derivation procedure is similar to the one specified in [RFC2716] section 3.5.

The client then proceeds to step 5.

If the **authentication** header field is for the Kerberos protocol or it does not contain a **gssapidata** parameter, the protocol client destroys the SA and proceeds to step 4.

4. If the protocol client does not find the matching SA or it has destroyed the SA in the preceding steps 2 or 3, the protocol client creates a new SA, and invokes a **GSS_Init_sec_context** call of the authentication protocol to initialize the security context.

5. For NTLM, the protocol client obtains user credentials, such as user name, password, and domain, and requests the following parameters, as specified in [MS-NLMP]:

- **Datagram**
- **Identify**
- **Integrity**

The NTLM implementation returns **NEGOTIATE_MESSAGE** as the output from the call to **GSS_Init_sec_context**. However, in the current NTLM implementation, this message is not generated for **datagram** NTLM contexts, and thus the output from NTLM is an empty buffer.

For Kerberos, the protocol client first requests a ticket to a service named in the **targetname** parameter value. It then creates the context by a call to **GSS_Init_sec_context** and requests the following parameters, as specified in [MS-KILE]:

- **Integrity**
- **Identify**

For TLS-DSK, the client initializes a context for performing TLS negotiation and generates a token with the **client_hello** handshake message, as specified in [RFC2246].

The protocol client stores generated context with the SA and accepts any token generated by the authentication (2) protocol to be encoded and sent to the server (2) in the following step.

6. The protocol client generates an **Authorization** header field if the challenge contained a **WWW-Authenticate** header field or a **Proxy-Authorization** header field if the challenge contained a **Proxy-Authenticate** header field, according to the syntax described in section 2.2.3, with the following data:

1. Authentication protocol ("NTLM", "Kerberos", or "TLS-DSK").

2. **realm** with the value copied from the challenge response.

3. **targetname** with the value copied from the challenge response.

4. **opaque** with the value copied from the challenge response if it was present.

5. **qop** with the value "auth".

6. **gssapi-data** with the value of the token generated in the preceding steps 3 or 4 and encoded using the base64 encoding procedure, as specified in [RFC3548] section 3. If the authentication protocol does not generate a token in step 3, such as in the case of the TLS-DSK protocol, the client does not add the **gssapi-data** parameter to the header field. However, if the authentication protocol generated an empty token in step 4, as in the case of the NTLM protocol, the **gssapi-data** parameter is added with an empty string as its value.

7. If the protocol client implements version 3 of this protocol, and the server challenge response carries a **version** parameter, and its value is 3 or higher or the protocol client implements version 4 of this protocol and the server response carries a **version** parameter, and its value is 3, the protocol client adds a **version** parameter with a value of 3.

8. If the protocol client implements version 4 of this protocol, and the server challenge response carries a **version** parameter and its value is 4 or higher, the protocol client adds a **version** parameter with a value of 4. The protocol client also generates **cnum**, **crand**, and **response** values, as described in section 3.2.4.1, steps 1, 2, and 3, and adds them to the header.

9. The protocol client resends the original request that was challenged by the server with the **Authorization** or **Proxy-Authorization** header created in step 5.

3.2.5.2 Processing Authenticated Messages from the SIP Server

When a SIP protocol client receives a message from the server that contains an **Authentication-Info** or **Proxy-Authentication-Info** header field, it MUST attempt to locate an existing security association that was created from the challenge that had the same authentication protocol, **realm**, and authentication target values as in the **Authentication-Info** or **Proxy-Authentication-Info** header field. For the NTLM and TLS-DSK<9> authentication protocols, the authentication target is the value of the **targetname** parameter without quotes. For the Kerberos authentication protocol, the authentication target is the value of the **targetname** parameter without quotes and the **sip/** service descriptor. The protocol client MUST then process the message as described in the following steps:

1. If the protocol client does not find the SA with a matching authentication protocol, **realm**, and authentication target values, it discards the message and stops further processing.

2. If the protocol client does find the SA with a matching authentication protocol, **realm**, and authentication (2) target values, the protocol client constructs a buffer with the information from the message that will be used in signature verification.

   The buffer is constructed from the following string values in order, each of them enclosed by angle brackets (<>), and with the same syntax and case, even if the field is case-insensitive, as they appear in the message headers:

   1. Authentication protocol ("NTLM", "Kerberos", or "TLS-DSK").

   2. **srand** value.

   3. **snum** value.

   4. **realm** parameter value without quotes.

5. **targetname** parameter value without quotes.
6. The value of the **Call-ID** header field.
7. The sequence number from the **CSeq** header field.
8. The method from the **CSeq** header field.
9. The URI in the **From** header field.
10. The **tag** parameter value from the **From** header field.
11. If the protocol version on the authenticating server is 3 or higher, the URI in the **To** header field.
12. The **tag** parameter value from the **To** header field.
13. If the protocol version on the authenticating server is 3 or higher, the **sip** URI from the **P-AssertedIdentity** header field.
14. If the protocol version on the authenticating server is 3 or higher, the **tel** URI from the **P-AssertedIdentity** header field.
15. The value of the **Expires** header field.
16. If the message is a response, the response code value as a decimal string.

If a field mentioned in the list does not exist in the message, an empty string enclosed by angle brackets is included. This does not apply to fields included conditionally depending on protocol version or message type; an empty string enclosed by angle brackets is not needed if the condition is not satisfied. However, empty angle brackets are included if the condition is satisfied, but there is no corresponding header field in the message.

3. The protocol client decodes the value of the **rspauth** parameter using the base16 decoding procedure, as specified in [RFC3548] section 6, and passes it, along with the buffer constructed in the preceding step 2, and the authentication protocol context from the SA to the **GSS_VerifyMic** call, as specified in [MS-NLMP] section 3.1.4 for NTLM, or [RFC2743] section 2.3.2 for Kerberos.

   Note that for the NTLM SSPI, the protocol client provides a fixed message sequence number of 100 in addition to the buffer.

   For TLS-DSK, the client computes the signature token using the HMAC algorithm specified in [RFC2104], with the hash function and server authentication key obtained when TLS negotiation completed, which means that the **finished** handshake message was received from the server, as described in section 3.2.5.1, and the buffer constructed in step 2. It then compares the binary value of the computed signature token with the decoded binary value of the **rspauth** parameter.

4. If the **GSS_VerifyMic** call fails, indicating that the signature could not be verified or the binary value of the computed TLS-DSK signature token is not the same as the binary value of the **rspauth** parameter, the protocol client discards the message and stops further processing.

5. If the **GSS_VerifyMic** call succeeds, the protocol client verifies that the sequence number in the **snum** parameter value does not fall outside the sliding window that it maintains and is not a replay of the message within the window. If the highest sequence number that the protocol client has processed so far for this SA exceeds the received sequence number by more than 256, or another message with the same sequence number has been received, the protocol client discards the message and stops processing.

6. If the sequence number is the highest seen so far for the SA, the protocol client adjusts the window and records the fact that this particular sequence number has been used.

7. If the SA used for verification is not yet in the "established" state, the protocol client determines whether the message it is processing is a **403 Forbidden** response. If the message is a 403 Forbidden response, the protocol client destroys the SA and requests different credentials from the user to retry the authentication.

   Otherwise, the protocol client transitions the SA to the "established" state and destroys any other SAs with the same **realm** and authentication target values.

3.2.5.3 Authenticated Address-Of-Record in Messages Signed By the SIP Server

If the SIP protocol client needs to obtain an authenticated identity, or address-of-record, from the message signed by the SIP server that the SIP client has verified, as described in [MS-NLMP] section 3.2.5.2, the protocol client MUST use the following procedure.

1. If the authenticating server protocol version is 3 or higher and the **P-AssertedIdentity** header field is present in the message, the protocol client uses the **sip** URI from the **P-AssertedIdentity** header field if it needs the SIP address-of-record, or uses the **tel** URI from the **P-AssertedIdentity** header field if it needs a telephone number.

2. If the authenticating server protocol version is less than 3 or if the **P-AssertedIdentity** header field is NOT present in the message, the protocol client uses the URI in the **From** header field if the message is a request, and uses the URI in the **To** header field if the message is a response.

   For more information about server protocol version, see [MS-NLMP] section 6.

3.2.5.4 Processing p-session-on-behalf-of Header in Messages from the SIP Server

This section follows the product behavior described in product behavior note <10>.

When a SIP protocol client receives an incoming request or a 2xx response with a **p-session-on-behalf-of header**, it MAY use this information to indicate to the end user that the message originator is acting on behalf of the address-of-record specified in the **p-session-on-behalf-of** header field.

For more information about setting up delegates, see [MS-PRES].

3.2.5.5 Responding as anonymous client to challenge from SIP Server

When the client receives a 401 Unauthorized challenge with digest and its **From:** header is an anonymous URI of the form <username>@anonymous.invalid and the destination is a conference GRUU, as specified in [MS-SIPRE], it SHOULD<11> check that algorithm is either **MD5-sess** or **SHA256-sess** and perform digest authentication, as specified in [RFC3261] section 22.4. The client SHOULD use the **conference-key** as the password for the computation of the digest for the **response** parameter of the **Authorization** or **Proxy-Authorization** header field. The protocol for obtaining the **conference-key** is specified in [MS-CONFBAS], with an example in [MS-CONFBAS] section 4.1. When using **SHA256sess** algorithm, the computation of the message digest in the response field SHOULD be performed as described in [FIPS180-2].

3.2.5.6 Continuing session as anonymous client

Once the SA is established between the anonymous client and the server, further messages from the client to the server (2) SHOULD<12> have the **Authorization** header constructed pre-emptively to reduce the round-trip cost associated with the challenge and response.

3.2.6 Timer Events

When the SA expiration timer fires, the SIP protocol client MUST initiate establishment of a new SA with the authenticating SIP server. The protocol client SHOULD send the REGISTER request without

authorization header fields, as described in section 3.1.3, to establish a new SA. The protocol client MAY use other requests, such as INVITE and SUBSCRIBE, also without header fields. The protocol client SHOULD NOT use ACK or CANCEL requests or other requests that the SIP protocol does not allow to challenge.

If establishment of a new SA completes successfully, the expired SA is destroyed, as described in section 3.2.5.2. Otherwise, the protocol client SHOULD keep the expired SA for the maximum duration of the **SIP transaction**. This timer is Timer B for INVITE transactions and Timer F for non-INVITE transactions, both 32 seconds by default, as described in [RFC3261].

3.2.7 Other Local Events

None.

3.3 SIP Server Details

3.3.1 Abstract Data Model

If the server is configured to authenticate the endpoints from which it receives messages, the server establishes and maintains a security association with each of these SIP client endpoints. SA establishment is initiated by the protocol client sending a REGISTER or other request, without authorization header fields, to the server.

The SA data includes:

- **Endpoint identification:** A **tuple** that consists of the address-of-record and one of the endpoint identifiers described in the Session Initiation Protocol Routing Extensions, as specified in [MS-SIPRE], such as the **epid** parameter in the **From** header field, the **+sip.instance** parameter in the **Contact** header field, or the GRUU in the **Contact** header field.
- **Client protocol version:** The value of the **version** parameter from the **authorization** header field of the protocol client's message that it sends in response to the challenge when the SA is created.
- **Server opaque value:** An arbitrary value that satisfies the syntax requirements specified in [RFC3261] section 25.1. It is generated by the server when it creates the SA.
- **State or phase:** The "establishing" state occurs during the NTLM, Kerberos, or TLS-DSK<13> authentication handshake. The "established" state occurs after NTLM, Kerberos, or TLS-DSK authentication completes and the server signs outgoing messages and verifies signatures on incoming messages.
- **Waiting for signature** flag: If the server implements version 4 of the authentication protocol, it sets this flag if the message with the authentication response from the protocol client does not carry a signature, and clears this flag when it receives a subsequent message from the protocol client with a valid signature.
- **SA expiration time:** The time when the SA expires and needs to be destroyed.
- **The SA idle time:** The time when the SA is destroyed if there is no message traffic from or to the client.
- **Outgoing message sequence number counter:** The counter that is incremented every time the message is sent with an **authentication information** header field created using this SA.
- **The high sequence number (SnumHigh):** The highest sequence number for messages received from the protocol client endpoint so far, and verified using this SA.

- **The list of received sequence numbers within sliding window:** All sequence numbers extracted from messages sent by the protocol client that fit into the sliding window between **SnumHigh-256** and **SnumHigh**.

- **The authentication protocol context:** The context information used by an authentication protocol compliant with GSS, and further clarified for NTLM in [MS-NLMP] and for Kerberos in [MS-KILE]. The context allows the authentication protocol to perform an authentication handshake when the SA is still in the "establishing" state, and to sign outgoing messages or verify the integrity of incoming messages using the attached signature when the SA enters the "established" state. For the TLS-DSK authentication protocol, when the SA is still in the "establishing" state, the server maintains enough context information to process TLS messages, as specified in [RFC2246]. Once the SA enters the "established" state, the server context maintains the hash function from the ciphersuite negotiated by TLS, and client and server authentication keys generated at the end of the TLS negotiation.

- **DelegateSet**: A set of delegate entries. Entries are keyed on the URI of the delegate. This is further defined in [MS-PRES].<14>

- **DelegateEntry**: Information about an individual delegate. This is further defined in [MS-PRES].<15>

  The server (2) maintains a table of SAs that it has established, indexed by protocol client endpoint (5) identity.

  The server (2) maintains a database that maps users identified through authentication (2) protocol processing to the address-of-records that these users are allowed to use.

  If the server is a proxy and it forwards requests after processing them with the SA, as described in section 3.3.4.1 and 3.3.5.2, it saves the reference to the SA in the **Via**, **Record-Route** or **Path** header fields that it inserts into the request, as specified in [MS-SIPRE] section 3.7, before forwarding the request to another **SIP element**. The content of the header fields is preserved within the transaction, dialog, or registration correspondingly, and the proxy recovers the reference to the SA that it saved, and use it to sign the messages it forwards to the protocol client endpoint (5) or to verify the signature in the messages it receives from the protocol client endpoint (5).

3.3.2 Timers

When the NTLM, Kerberos, or TLS-DSK<16> authentication handshake completes and the SA enters the "established" state, the SIP server MUST start an SA expiration timer with a value of 8 hours.

When the server finishes processing a message that it sends to the protocol client, as described in section 3.3.4.1, or when it successfully finishes authenticating or validating a signature in a message from the protocol client, as described in sections 3.3.5.2 or 3.3.5.3, the server (2) SHOULD start an idle timer or reset the idle timer, if it is already running, with a value chosen based on the following criteria evaluated in order:

1. If the message being sent to the protocol client is a 2xx response to the REGISTER request and it contains an **Expires** header field, the server SHOULD extract the **delta-seconds** value from the **Expires** header field and use it as the timer value in seconds.

2. If the message being sent to the protocol client is a 2xx response to the INVITE or UPDATE request and it contains a **Session-Expires** header field, as specified in [RFC4028], and either the idle timer is not already running or its last value was not set from the **Expires** header field of the 2xx response to the REGISTER request, as described previously, the server SHOULD extract the **delta-seconds** value from the **Session-Expires** header field and use it as the timer value in seconds.

3. If the message is received from the protocol client and either the idle timer is not already running or its last value was not set from the **Expires** or **Session-Expires** header fields as described previously, the server SHOULD set the timer value to 900 seconds, or 15 minutes.

3.3.3 Initialization

The SIP server MUST select **realm** and **targetname** values that uniquely identify it among all the possible principals (1) within the NTLM, Kerberos, and TLS-DSK<17> authentication protocol namespaces. The exact meaning of **realm** is specified in [RFC3261] section 22.1. The **targetname** corresponds to the **TargetName** in NTLM, as specified in [MS-NLMP] and **principal name** for Kerberos, as specified in [RFC4120]. For Kerberos, the **targetname** value MUST be prefixed with a **sip/** service descriptor and the server MUST register its **targetname** value with the KDC database. For TLS-DSK, **targetname** MUST match the value of the **subjectAltName** extension of type "dNSName", if one is present, or the **Common Name** field in the **Subject** field of the server certificate, as specified in [RFC2818] section 3.1. If the server supports more than one authentication protocol, the **targetname** value in NTLM and TLS-DSK MUST be the same as the **targetname** value in Kerberos without the **sip/** service descriptor. The server SHOULD use its **fully qualified domain name (FQDN)** as the **targetname** value for NTLM and TLS-DSK, and its FQDN prefixed with a **sip/** service descriptor for Kerberos.

3.3.4 Higher-Layer Triggered Events

3.3.4.1 Sending Messages to the SIP Client

When the SIP server needs to send a message to a SIP protocol client endpoint, it MUST check for security associations that were established with that endpoint. The server SHOULD use an SA reference that it has previously placed into the **Via**, **Record-Route**, or **Path** header field of the associated message. This could be a request that initiated the transaction if the message being processed is a response, a request that initiated the **dialog** if the message being processed is a mid-dialog request, or a REGISTER request if the message being processed is a request that is delivered along the registration path. If there is an SA in the "established" state, the server MUST use this SA to generate and insert an **authorization** header field using the following steps:

1. If the server implements version 4 of the authentication protocol and the **waiting for signature** flag in the SA is set, and the message being processed is a request, the server MUST reject the request with 500 Server Internal response and stop further processing.

2. The server (2) increments its outgoing message sequence number counter and generates a server (2) random value.

   The server (2) outgoing sequence number (**snum**) is maintained on a per-SA basis and incremented each time this procedure is performed. It is stored as an unsigned decimal number, as described in ABNF in section 2.2.2.

   A server (2) random value (**srand**) is a 32-bit nonce. It is stored as an eight-digit hexadecimal number, as described in ABNF in section 2.2.2.

   The server (2) constructs a buffer with the information from the message and the SA that will be used in the signature computation.

3. The buffer is constructed from the following string values encoded in UTF8 in order, each of them enclosed by angle brackets (<>), and with the same syntax and case, even if the field is case-insensitive, as they appear in the message header fields:

   1. Authentication protocol ("NTLM", "Kerberos", or "TLS-DSK").

   2. **srand** value as an eight-digit hexadecimal number.

3. **snum** value as a decimal number.
4. **realm** with the value selected by the server during initialization.
5. **targetname** with the value selected by the server during initialization.
6. The value of the **Call-ID** header field from the message.
7. The sequence number from the **CSeq** header field.
8. The method from the **CSeq** header field.
9. The URI in the **From** header field.
10. The **tag** parameter value from the **From** header field.
11. If the authenticating server protocol version is 3 or higher, the URI in the **To** header field.
12. The **tag** parameter value from the **To** header field.
13. If the protocol client protocol version stored in the SA is 3 or higher, the **sip** URI from the **P-Asserted-Identity** header field.
14. If the protocol client protocol version stored in the SA is 3 or higher, the **tel** URI from the **P-Asserted-Identity** header field.
15. The value of the **Expires** header field.
16. If the message is a response, the response code value as a decimal string.

    If any field mentioned in the list does not exist in the message, an empty string enclosed by angle brackets is included. This does not apply to fields included conditionally depending on protocol version or message type; an empty string enclosed by angle brackets is not needed if the condition is not satisfied. However, empty angle brackets are included if the condition is satisfied, but there is no corresponding header field in the message.

4. The server uses an authentication protocol **GSS_GetMIC()** call, as described in [MS-NLMP] section 3.1.4 for NTLM**,** or in [RFC2743] section 2.3.1 for Kerberos, to generate a signature token for the buffer constructed in step 3, using the authentication protocol context stored in the SA.

   Note that for the NTLM SSPI, the server (2) provides a fixed message sequence number of 100, in addition to the buffer and protocol context.

   For TLS-DSK<18>, the server computes the signature token using the HMAC algorithm specified in [RFC2104], with the hash function and server authentication key obtained when TLS negotiation completed, which means the **finished** handshake message was received from the client, as described in section 3.3.5.1, and the buffer constructed in step 3.

   The binary token returned by the authentication protocol implementation is then encoded using the Base16 encoding procedure specified in [RFC3548] section 6. The characters 'A' through 'F' in the output of the Base16 encoding procedure SHOULD be replaced with their lowercase equivalents ('a' through 'f').

5. The server generates an **Authentication-Info** header field, if it acted as a UAS when the SA was established, or a **Proxy-Authentication-Info** header field, if it acted as a proxy when the SA was established, according to the syntax described in section 2.2.2, with the data generated in the preceding steps and data generated during initialization or SA creation**.**

   Specifically, it MUST add the following fields:

   1. Authentication protocol ("NTLM", "Kerberos", or "TLS-DSK").

2. **realm** with the value selected by the server during initialization.
3. **targetname** with the value selected by the server during initialization.
4. **opaque** with the value that the server generated during SA creation.
5. **qop** with the value "auth".
6. **snum** with the value generated in step 2.
7. **srand** with the value generated in step 2.
8. **rspauth** with the value generated in step 4.
9. **version** with the value of 3 if the server implements version 3 of this protocol or 4 if the server implements version 4 of this protocol

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Processing Unauthenticated Messages from the SIP Client

When a SIP server that is configured to authenticate all SIP protocol clients that talk to it receives a message from the protocol client that does not carry an either an **Authorization** or **Proxy-Authorization** header field, or the **realm** and **targetname** parameter value pairs in all of the **authorization** header fields do not match the values that the server created during initialization, the server MUST reject the message. If the message is not an ACK or CANCEL, the server MUST send a 401 Unauthorized or 407 Proxy Authentication Required response, or challenge, back to the protocol client with one or more **authentication** header fields. If the message is an ACK or CANCEL, the server MUST discard it.

When forming a challenge response, the server SHOULD add an **authentication** header field for each authentication protocol that it supports, such as NTLM, Kerberos, and TLS-DSK<19> **authentication** header fields that are covered in this protocol, with the following content:

- Authentication protocol ("NTLM", "Kerberos", or "TLS-DSK").
- **realm** with the value selected by the server during initialization.
- **targetname** with the value selected by the server during initialization.
- **version** with the value of 3<20> if the server implements version 3 of this protocol or 4<21> if the server implements version 4 of this protocol.
- For **TLS-DSK** authentication protocol the server SHOULD add an **sts-uri** parameter with the value of the URL of the Certificate Provisioning Service described in [MS-OCAUTHWS].

  However, if the request is destined to a conference GRUU, as specified in [MS-SIPRE] and the **From:** header in the request is an anonymous URI of the form <username>@anonymous.invalid, the server SHOULD<22> form the challenge response as described in section 3.3.5.4 and the rest of this section SHOULD NOT be followed.

  The server SHOULD use a 401 Unauthorized response with a **WWW-Authenticate** header field if it acts as a UAS when processing the request, and it SHOULD use a 407 Proxy Authentication Required response with a **Proxy-Authenticate** header field if it acts as a proxy when processing the request.

  The server MUST add a **Date** header field with the value obtained from the computer or device on which it runs.

3.3.5.2 Processing Messages with Authentication Response from the SIP Client

When the SIP server receives a request from the SIP protocol client that carries either an **Authorization** or **Proxy-Authorization** header field and the **realm** and **targetname** parameter values in this header field match the values that the server created during initialization, and the **gssapi-data** parameter is present, the server MUST perform the following steps:

1. Extract all endpoint identifiers for the UAC endpoint that are present in the request. The server compliant with this specification MUST use the following UAC endpoint identifiers, as specified in [MS-SIPRE]:

    1. Address-of-record from the **From** header field and the **epid** parameter value from the **From** header field.

    2. Address-of-record from the **From** header field and the **+sip.instance** parameter value from the **Contact** header field.

    3. Address-of-record from the **From** header field and the GRUU from the **Contact** header field.

    A message can have one or more endpoint identifiers. If more than one endpoint identifier is present in the request, the server MUST validate that the **+sip.instance** value was properly derived from the **epid** value, as specified in [MS-SIPRE], and that the GRUU was generated by the **SIP registrar** for the endpoint identified with either the **epid** or the **+sip.instance** values, as specified in [MS-SIPRE].

2. If the **authorization** header field is for the NTLM or TLS-DSK<23> authentication protocols, which require more than one round trip to complete the handshake, the server MUST attempt to locate the SA that it already created during the first round-trip of the negotiation.

    To locate the SA, the server SHOULD use the **opaque** parameter value from the **authorization** header field in the request along with the UAC endpoint identifier extracted from the request in step 1**.** If the server locates such an SA, it MUST skip step 3 and proceed directly to step 4.

3. If the **authorization** header field is for the Kerberos authentication protocol, or the server could not locate the SA for the NTLM or TLS-DSK protocol in the previous step, it MUST create the SA and capture the authentication protocol and the identity of the UAC endpoint in it. The server MUST also capture the **version** parameter value from the **authorization** header field in the request. If the **version** parameter does not exist, the server MUST conclude that the protocol client uses a version number of 2. The server SHOULD generate the value for the **opaque** parameter and capture it in the SA.

4. The server MUST then decode the value of the **gssapi-data** parameter from the **authorization** header field using the base64 encoding defined in [RFC3548] section 3, and use it in the **GSS_accept_security_context** call, as specified in [RFC2743] for Kerberos, or pass it down to the NTLM implementation as **NEGOTIATE_MESSAGE** if this is the first round-trip of the NTLM handshake, or as **AUTHENTICATE_MESSAGE** if it is the second round-trip, as specified in [MS-NLMP]. When creating a security context for the Kerberos authentication protocol, the server MUST use the **Mutual Authentication**, **Integrity**, and **Identify** parameters, as specified in [MS-KILE]. When creating a security context for the NTLM authentication protocol, the server MUST use the **Datagram**, **Identify**, and **Integrity** parameters, as specified in [MS-NLMP]. For TLS-DSK, the server MUST pass the decoded value of the **gssapi-data** parameter to the TLS implementation for processing, as specified in [RFC2246]. During the first round-trip, the value contains the TLS **client_hello** handshake message, while during the second round-trip the value carries the TLS **certificate**, **client_key_exchange**, **certificate_verify**, **change_cipher_spec**, and **finished** handshake messages, and the third round-trip does not have the **gssapi-data** parameter.

5. If the authentication protocol processing in step 4 failed, indicating that the protocol client could not be authenticated, and the **authorization** header field is for the Kerberos authentication

protocol, or it is for NTLM and this is the second round-trip, which means that the SA already existed when the request arrived and it was located in step 2, or it is for TLS-DSK and this is the second or the third round trip, which means that the SA already existed when the request arrived and it was located in step 2, the server MUST reject the request with a 401 Unauthorized or 407 Proxy Authentication Required response and stop further processing, as described in section 3.2.5.1, as though the request did not have the **authorization** header field.

6. If authentication protocol processing in step 4 succeeded and returned no output data, the server MUST proceed directly to the next step . If the authentication protocol implementation produced output data, the server MUST process the output of the authentication protocol and send it back to the protocol client using a 401 Unauthorized response with a **WWW-Authenticate** header field if the server is acting as a UAS, and a 407 Proxy Authentication Required response with a **Proxy-Authenticate** header field if the server is acting as a proxy as follows:

    1. If the **authorization** header field is for the NTLM authentication protocol and this is the first round-trip and the SA was created in step 3, the NTLM authentication protocol implementation on the server generates the **CHALLENGE_MESSAGE**.If the **authorization** header field is for the TLS-DSK authentication protocol and this is the first round trip, the TLS implementation on the server generates a response that encapsulates one or more TLS records that contain a TLS **`server_hello`** handshake message, followed by a TLS certificate, such as **server_key_exchange**, and then the **`certificate_request`** and **`server_hello_done`** handshake messages, as specified in [RFC2246]. If the **authorization** header field is for the TLS-DSK authentication protocol and it is the second round trip, the TLS implementation on the server generates a response that encapsulates one or more TLS records that contain TLS **`change_cipher_spec`** and **`finished`** handshake messages. At this point the server MUST note the hash function from the ciphersuite selected by the TLS negotiation and compute the server and client authentication keys as follows (the key derivation procedure that follows is similar to the one described in [RFC2716] section 3.5):

        1. Given the master secret negotiated by the TLS handshake, the pseudo-random function (PRF) defined in the specification for the version of TLS in use, and the value randomly defined as the concatenation of the handshake message fields **client_hello.random** and **server_hello.random**, in that order, the value PRF (master secret, "client EAP encryption", random) is computed up to 128 bytes.

        2. The client authentication (2) key, which is the one used for computing and validating signatures for messages from client to server (2), is obtained by truncating to the correct length the third 32 bytes of the PRF output string.

        3. The server (2) authentication (2) key, which is the one used for computing and validating signatures for messages from server (2) to client, is obtained by truncating to the correct length the fourth 32 bytes of this same PRF output string.

    2. The server MUST encode the **CHALLENGE_MESSAGE** returned by the NTLM implementation or the preceding TLS responses, using the base64 encoding described in [RFC3548] section 3, and populate the **WWW-Authenticate** or **Proxy-Authenticate** header field with the following parameters:

        - Authentication protocol ("NTLM" or "TLS-DSK").
        - **realm** with the value selected by the server during initialization.
        - **targetname** with the value selected by the server during initialization.
        - **opaque** with the value that the server generated during SA creation.
        - **version** with a value of 3 or 4, depending on the version of the authentication protocol that the server supports.

- **gssapi-data** with the base64 encoded **CHALLENGE_MESSAGE** from NTLM or the response from TLS.

3. After sending the 401 or 407 response with the data described in the previous step, the server (2) MUST discard the request and stop further processing.

7. If the server implements version 4 of the authentication protocol, and the **authorization** header has a **version** parameter with a value of 4 or higher, the server MUST check if the **authorization** header also contains **response**, **crand**, and **cnum** parameters. If any of these parameters are not present, the server MUST reject the request with a 401 Unauthorized or 407 Proxy Authentication Required response and stop further processing, as described in section 3.2.5.1, as though the request did not have the **authorization** header field.

8. If the server implements version 4 of the authentication protocol, it MUST check if the **authorization** header contains the **response**, **crand**, and **cnum** parameters. These parameters MUST be present if the protocol client also implements version 4 of the authentication protocol and are enforced by the check in the previous step. However, they might also be present in the responses generated by the protocol client implementing a lower version of the authentication protocol.

    1. If all of the parameters are present, the server MUST perform signature validation, as described in section 3.3.5.3 steps 2, 3, 4, and 5. If one of the steps failed, the server MUST stop further processing. Otherwise it MUST proceed to step 9.

    2. If not all of the parameters are present and the server already has an SA in "established" state and not marked as **waiting for signature** with the same protocol client endpoint connected from the same transport address, the server MUST proceed to the step 9 as though it has successfully validated the signature.

    3. If not all of the parameters are present and message being processed is:

        1. A REGISTER request with the **Expires** header field value greater than 0, or

        2. An INVITE request with a URI in the **To** header field that conforms to the ABNF of **conf-endpoint-gruu**, as specified in [MS-SIPRE] section 2, or

        3. A SUBSCRIBE request with the **Event** header field value of "vndmicrosoftprovisioningv2" and the **Content-Type** header field value of "application/vnd-microsoft-roaming-provisioning-v2+xml", the server MUST set the **waiting for signature** flag in the SA and proceed to step 9.

    4. Otherwise, the server MUST reject the request with a 401 Unauthorized or 407 Proxy Authentication Required response and stop further processing, as described in section 3.3.5.1, as though the request did not have the **authorization** header field.

9. If the authentication processing in the preceding steps succeeded, the server MUST determine whether the user authenticated by the NTLM, Kerberos, or TLS-DSK protocol is authorized to use the address-of-record in the URI of the **From** header field of the request.

    If an authenticated user is not authorized, the server MUST reject the request with a 403 Forbidden response. The server MUST add an **authentication information** header field to the 403 Forbidden response, as described in section 3.3.4.1, using the SA located in step 2 or created in step 3. After a 403 Forbidden response is sent, the server (2) MUST destroy the SA and stop further processing.

10. If the authorization in step 9 succeeded, the server (2) MUST transition the SA into the "established" state. The server (2) SHOULD then continue processing the request as required by the SIP protocol.

3.3.5.3 Processing Authorized Messages from the SIP Client

When the SIP server receives a message from a SIP protocol client endpoint that contains an **authorization** header field, which is either an **Authorization** or **Proxy-Authorization** header field, with a **response** parameter, and the **realm** and **targetname** parameter values in the **authorization** header field match the values that the server created during initialization, the server MUST perform the following steps.

1. The server attempts to locate an existing security association that it previously created for the protocol client endpoint that matches the authentication protocol in the **authorization** header. The server uses one or a combination of the following methods to identify the protocol client endpoint:

    1. The reference to the SA that it placed into the **Via** or **RecordRoute** header fields in previous messages in the same transaction or dialog with the same endpoint.

    2. The **opaque** parameter that it previously placed into the authentication header fields of the previous messages with the same endpoint.

    3. The endpoint identifier, as specified in [MS-SIPRE].

        If the server cannot locate the SA, or the SA it located is not in the "established" state, the server MUST reject the message and respond with a 401 Unauthorized or 407 Proxy Authentication Required response code if the message was a request, and stop further processing, as described in section 3.3.5.1, as though the message did not have the **authorization** header field.

2. Otherwise, the server (2) constructs a buffer with the information from the message that will be used in signature verification. The buffer is constructed from the following string values in order, each of them enclosed by angle brackets (<>), with the same syntax and case, even if the field is case-insensitive, as they appear in the message headers:

    1. Authentication protocol ("NTLM", "Kerberos", or "TLS-DSK").

    2. **crand** value.

    3. **cnum** value.

    4. **realm** parameter value without quotes.

    5. **targetname** parameter value without quotes.

    6. The value of the **Call-ID** header field.

    7. The sequence number from the **CSeq** header field.

    8. The method from the **CSeq** header field.

    9. The URI in the **From** header field.

    10. The **tag** parameter value from the **From** header field.

    11. If the protocol version of the protocol client captured in the SA is 3 or higher, the URI in the **To** header field.

    12. The **tag** parameter value from the **To** header field.

    13. If the protocol version of the protocol client captured in the SA is 3 or higher, the **sip** URI from the **PAssertedIdentity** or **P-Preferred-Identity** header field.

14. If the protocol version of the protocol client captured in the SA is 3 or higher, the **tel** URI from the **PAssertedIdentity** or **P-Preferred-Identity** header field.

15. The value of the **Expires** header field.

16. If the message is a response, the response code value as a decimal string.

    If any field mentioned in the list does not exist in the message, an empty string enclosed by angle brackets is included. This does not apply to fields included conditionally, depending on protocol version or message type. An empty string enclosed by the angle brackets is not needed if the condition is not satisfied; however, empty angle brackets are included if the condition is satisfied, but there is no corresponding header field in the message.

3. The server decodes the value of the **response** parameter, using the **base16** decoding procedure as specified in [RFC3548] section 6, and passes it along with the buffer constructed in step 2, and the authentication protocol context from the SA to the **GSS_VerifyMic** call, as described in [MS-NLMP] section 3.1.4 for NTLM, or [RFC2743] section 2.3.2 for Kerberos.

   Note that for the NTLM SSPI, the server (2) provides a fixed message sequence number of 100 in addition to the buffer.

   For TLS-DSK<24>, the server computes the signature token using the HMAC algorithm specified in [RFC2104] with the hash function and client authentication key obtained when TLS negotiation completed, which means that the **finished** handshake message was received from the client, as described in section 3.3.5.1, and the buffer constructed in step 2. It then compares the binary value of the computed signature token with the decoded binary value of the **response** parameter.

4. If the **GSS_VerifyMic** call fails, indicating that the signature could not be verified, or the binary value of the computed TLS-DSK signature token is not the same as the binary value of the **response** parameter, the server rejects the message and responds with a 401 Unauthorized or 407 Proxy Authentication Required response code if the message was a request, and stops further processing, as described in section 3.3.5.1, as though the message did not have the **authorization** header field.

5. The server then verifies that the sequence number in the **cnum** parameter value does not fall outside the sliding window that it maintains and is not a replay of the message within the window. If the highest sequence number that the server has processed so far for this SA exceeds the received sequence number by more than 256, or if another message with the same sequence number has been received, the server rejects the message and responds with a 401 Unauthorized or 407 Proxy Authentication Required response code if the message was a request, and stops further processing, as described in section 3.3.5.1, as though the message did not have the **authorization** header field.

   Otherwise, the server (2) adjusts the window if the sequence number is the highest seen so far for the SA, and records the fact that this particular sequence number has already been used.

6. If the server implements version 4 of the authentication protocol, and it has previously set the **waiting for signature** flag in the SA, it clears this flag.

7. The server removes the **authorization** header field from the message and continues processing the message as required by the SIP protocol.

3.3.5.4 Establishing session with anonymous client

When the server receives a request that is destined to a conference GRUU, as specified in [MS-SIPRE] and the **From:** header in the message is an anonymous URI of the form <username>@anonymous.invalid and the request does not carry either an **Authorization** or **Proxy-**

Authorization header field, it SHOULD<25> challenge with **Digest** authentication protocol as described in [RFC3261] section 22.4 with algorithm parameter value of **MD5-sess** or **SHA256-sess**. For an example, see section 4.5.

3.3.5.5 Processing Authorized Messages from anonymous client

This section follows the product behavior described in endnote<26>.

When the server receives a request that is destined to a conference GRUU, as specified in [MS-SIPRE], and the **From:** header in the message is an anonymous URI of the form <username>@anonymous.invalid and the message contains an **authorization** header field, which is either an **authorization** or **proxy-authorization** header field, it SHOULD validate that the authorization header field uses **digest** as authentication protocol and the algorithm parameter is either **MD5sess** or **SHA256sess**. If validation fails, the server SHOULD reject the request and respond with a 401 Unauthorized or 407 Proxy Authentication Required response code and stop further processing, as described in section 3.3.5.4, as though the request did not have the **authorization** header field.

If validation succeeds, the server SHOULD perform validation of the **authorization** header with **digest** authentication protocol as described in [RFC3261] section 22.4. The server SHOULD use the conference PIN as the password for the computation of the digest for the **response** parameter of the **authorization** or **proxy-authorization** header field. When using the **SHA256sess** algorithm, the computation of the message digest in the response field SHOULD be performed as described in [FIPS180-2].

3.3.5.6 Processing Alternate Identities in Messages from the SIP Client

The SIP server MUST process **P-Asserted-Identity** and **P-Preferred-Identity** header field values in messages that it receives from the SIP protocol client as described in [RFC3325] sections 5, 6, and 7. For the purposes of this specification, the server MUST consider any protocol client to be a node that it does not trust. It MUST consider other servers in the same domain as nodes that it trusts, and servers in other domains as nodes that it does not trust.

3.3.5.7 Processing p-session-on-behalf-of Header in Messages from the SIP Client

This section follows the product behavior described in product behavior note <27>.

When the authenticating SIP server receives a message that contains a **p-session-on-behalf-of** header, it MUST verify that the URI of the sender is a valid delegate by ensuring that it matches a **DelegateEntry** in the **DelegateSet** of the delegator, identified by the URI in the **p-session-on-behalf-of** header field.

If a matching **DelegateEntry** is found and the message is an INVITE request and the message does not have a **Referred-By** header field, the SIP server MUST insert a **referred-by** header with a **Referrer-uri** value set to the delegator's address-of-record. If no matching **DelegateEntry** is found, the SIP server MUST reject the message with a 403 Forbidden response if the message is a request. If no matching **DelegateEntry** is found and the message is an ACK or CANCEL, the SIP server MAY discard the message and stop processing.

The SIP server SHOULD reject a request message by returning a 400 response when the request message has a **p-session-on-behalf-of** header field value that contains URI or header parameters. The SIP server SHOULD discard and stop processing a response message if the message is an ACK or CANCEL response.

For more information about setting up delegates, see [MS-PRES].

3.3.6 Timer Events

When the SA expiration timer fires, the SIP server MUST discard the SA.

When the SA idle timer fires, the server (2) SHOULD discard the SA.

3.3.7 Other Local Events

None.

4 Protocol Examples

4.1 NTLM Authentication Example

1. Alice's SIP protocol client sends a REGISTER request with no **authorization** header field to the SIP server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4320
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 169 REGISTER
Contact: <sip:192.0.2.1:4320;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:4233FD41-
093B-5FD6-B5D2-651ED55969E6>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

2. Authentication is enabled at the server, which then challenges Alice's protocol client. The server indicates support for NTLM and Kerberos in the challenge and returns the **realm** and **targetname** values that it created during initialization, the version of the authentication protocol that it implements, and the **Date** header field.

```
SIP/2.0 401 Unauthorized
Date: Thu, 31 Jan 2008 00:01:56 GMT
WWW-Authenticate: NTLM realm="SIP Communications Service", targetname="server.contoso.com
", version=3
WWW-Authenticate: Kerberos realm="SIP Communications Service", targetname="sip/server.con
toso.com", version=3
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>;tag=0858513FA91D3AAE1A5840DDB99599DF
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 169 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4320;ms-received-cid=1C500
Content-Length: 0
```

3. The protocol client decides to use NTLM and creates an SA with data from the **authentication** header, specifically, **NTLM**, **realm**, **targetname**, and **version**. It calls the NTLM authentication protocol implementation with Alice's credentials (user name, domain, and password) and **Datagram**, **Identify**, and **Integrity** parameters, to initialize the security context and generate **NEGOTIATE_MESSAGE**. In the current NTLM implementation, this message is not generated for datagram NTLM contexts, so the output from NTLM is an empty buffer. Thus, the protocol client generates an **authorization** header field and sends the following request to the server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4320
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 170 REGISTER
Authorization: NTLM qop="auth", realm="SIP Communications Service", targetname="server.co
ntoso.com", gssapi-data="", version=3
Contact: <sip:192.0.2.1:4320;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:4233FD
41-093B-5FD6-B5D2-651ED55969E6>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

4. The server extracts the protocol client endpoint identity as the address-of-record in the **From** header field ("alice@contoso.com") and either the value of the **epid** parameter in the **From** header field ("8248ca9ebb") or the value of the **+sip.instance** parameter in the **Contact** header field ("urn:uuid:4233FD41093B5FD6B5D2651ED55969E6"). It creates the SA for the NTLM protocol, initializing it with the protocol client endpoint identifier, version (3), generated opaque value ("BCDC0C9D"), and calls into the NTLM implementation to process an empty **NEGOTIATE_MESSAGE**. The NTLM implementation creates the security context and generates a **CHALLENGE_MESSAGE** token that the server encodes and sends back to the protocol client in the following 401 Unauthorized response.

```
SIP/2.0 401 Unauthorized
Date: Thu, 31 Jan 2008 00:01:56 GMT
WWW-Authenticate: NTLM opaque="BCDC0C9D", gssapi-
data="12345678ABCDEF", targetname="server.contoso.com", realm="SIP Communications Service
", version=3
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>;tag=0858513FA91D3AAE1A5840DDB99599DF
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 170 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4320;ms-received-cid=1C500
Content-Length: 0
```

5. The protocol client locates the SA which it created for the first challenge message from the server, decodes the value of the **gssapi-data** parameter using the base64 algorithm, and passes it along as the security context information stored in the SA down to the NTLM implementation as **CHALLENGE_MESSAGE**. The NTLM implementation generates **AUTHENTICATE_MESSAGE**, which the protocol client encodes using the base64 algorithm, generates the **authorization** header field, and sends the following request to the server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4320
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 171 REGISTER
Authorization: NTLM opaque="BCDC0C9D", qop="auth", realm="SIP Communications Service", ta
rgetname="server.contoso.com", gssapi-data="12345678ABCDE", version=3
Contact: <sip:192.0.2.1:4320;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:4233FD
41-093B-5FD6-B5D2-651ED55969E6>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

6. The server extracts the protocol client endpoint identity from the request and the **opaque** value from the **authorization** header field, finds the existing SA and calls into the NTLM implementation to process **AUTHENTICATE_MESSAGE**. The NTLM implementation authenticates the protocol client. The server then extracts the user identity from the authentication protocol context and validates that the user is authorized to use the "alice@contoso.com" address-of-record. The server can now continue processing the REGISTER request, as described in [RFC3261], and pass it to the SIP registrar component.

7. After the SIP registrar component completes processing, it sends back a 200 OK response to the protocol client, which is processed by the authentication component on the server. This component locates the SA based on the reference it stored in the **Via** header field, or some other mechanism, and calls into the NTLM authentication protocol implementation to generate a signature token for the response. Based on the contents of the response shown in the following example, the server constructs the following buffer for the **GSS_GetMic()** call to NTLM.

```
<NTLM><0B9D33A2><1><SIP Communications Service><server.contoso.com><d5f2b95d5be64c2cbfb38
aa5d3a87ae7><171><REGISTER><sip:alice@contoso.com><4a2b44d131><sip:alice@contoso.com><085
8513FA91D3AAE1A5840DDB99599DF><><><7200><200>
```

8. NTLM returns the signature, and the server creates an **authentication information** header field and sends the following message to the protocol client.

```
SIP/2.0 200 OK
Authentication-Info: NTLM rspauth="01000000000000005CD422F0C750C7C6", srand="0B9D33A2", s
num="1", opaque="BCDC0C9D", qop="auth", targetname="server.contoso.com", realm="SIP Commu
nications Service"
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>;tag=0858513FA91D3AAE1A5840DDB99599DF
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 171 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4320;ms-received-cid=1C500
Contact: <sip:192.0.2.1:4320;transport=tls;msreceivedcid=1C500>;expires=7200;+sip.instanc
e="<urn:uuid:4233fd41093b5fd6b5d2651ed55969e6>";gruu="sip:alice@contoso.com;opaque=user:e
pid:Qf0zQjsJ1l10mUe1Vlp5gAA;gruu"
Expires: 7200
Content-Length: 0
```

9. The protocol client receives the message, locates the SA, and uses it to call into the NTLM implementation to verify the signature using its **GSS_VerifyMIC** call. The buffer that the protocol client creates for signature verification is identical to the buffer created by the server when it generated the signature.

4.2 Kerberos Authentication Example

1. Alice's SIP protocol client sends a REGISTER request with no **authorization** header field to the SIP server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4849
From: <sip:alice@contoso.com>;tag=604168c9c0;epid=2ebb6f264f
To: <sip:alice@contoso.com>
Call-ID: c7142b90f8c94668807a382f552a6770
CSeq: 1 REGISTER
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841E4-
264D-52E8-96C5-D22AA8CDC316>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

2. Authentication is enabled at the server, which then challenges Alice's protocol client. The server indicates support for NTLM and Kerberos in the challenge and returns the **realm** and **targetname** values that it created during initialization, the version of the authentication protocol that it implements, and the **Date** header field.

```
SIP/2.0 401 Unauthorized
Date: Sun, 16 Dec 2007 05:11:19 GMT
WWW-Authenticate: NTLM realm="SIP Communications Service", targetname="server.contoso.com
", version=3
WWW-Authenticate: Kerberos realm="SIP Communications Service", targetname="sip/server.con
toso.com", version=3
From: <sip:alice@contoso.com>;tag=604168c9c0;epid=2ebb6f264f
To: <sip:alice@contoso.com>;tag=9588410E2DA11CEE9D0AE7733E07830F
Call-ID: c7142b90f8c94668807a382f552a6770
CSeq: 1 REGISTER
```

```
Via: SIP/2.0/TLS 192.0.2.1:4849;ms-received-cid=900
Content-Length: 0
```

3. The protocol client decides to use Kerberos and creates an SA with data from the **authentication** header field, specifically, **Kerberos**, **realm**, **targetname**, and **version**. It obtains a Kerberos ticket for the service principal (2) in the **targetname** parameter ("sip/server.contoso.com"), and calls the Kerberos authentication protocol implementation with it and the **Identify** and **Integrity** parameters to initialize the security context and generate a KRB_AP_REQ token. The protocol client encodes the Kerberos token, using the base64 algorithms, and sends the following request to the server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4849
From: <sip:alice@contoso.com>;tag=604168c9c0;epid=2ebb6f264f
To: <sip:alice@contoso.com>
Call-ID: c7142b90f8c94668807a382f552a6770
CSeq: 2 REGISTER
Authorization: Kerberos qop="auth", realm="SIP Communications Service", targetname="sip/s
erver.contoso.com", gssapi-data="1234ABCDEF", version=3
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841
E4-264D-52E8-96C5-D22AA8CDC316>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

4. The server extracts the protocol client endpoint identity as the address-of-record in the **From** header field ("alice@contoso.com") and either the value of the **epid** parameter in the **From** header field ("2ebb6f264f") or the value of the **+sip.instance** parameter in the **Contact** header field ("urn:uuid:124841E4-264D-52E8-96C5-D22AA8CDC316"). It creates the SA for the Kerberos protocol, initializing it with a protocol client endpoint identifier, version, the generated **opaque** value ("A9A0BB9C"), and calls into the Kerberos implementation to initialize the security context and process the **KRB_AP_REQ** token. The Kerberos implementation authenticates the protocol client. The server then extracts the user identity from the authentication protocol context and validates that the user is authorized to use the "alice@contoso.com" address-of-record. The server can now continue processing the REGISTER request, as described in [RFC3261], and pass it to the SIP registrar component.

5. After the SIP registrar component completes processing, it sends back a 200 OK response to the protocol client, which is processed by the authentication component on the server. The component locates the SA based on the reference it stored in the **Via** header field, or on some other mechanism, and calls into the Kerberos authentication protocol implementation to generate a signature token for the response. Based on the content of the following response, the server constructs the following buffer for the **GSS_GetMic()** call to Kerberos.

```
<Kerberos><211639C4><1><SIP Communications Service><sip/server.contoso.com><c7142b90f8c94
668807a382f552a6770><2><REGISTER><sip:alice@contoso.com><604168c9c0><sip:alice@contoso.co
m><9588410E2DA11CEE9D0AE7733E07830F><><><7200><200>
```

6. Kerberos returns the signature, and the server creates an **authentication information** header field and sends the following message to the protocol client.

```
SIP/2.0 200 OK
Authentication-Info: Kerberos rspauth="602306092", srand="211639C4", snum="1", opaque="A9
A0BB9C", qop="auth", targetname="sip/server.contoso.com", realm="SIP Communications Servi
ce"
From: <sip:alice@contoso.com>;tag=604168c9c0;epid=2ebb6f264f
To: <sip:alice@contoso.com>;tag=9588410E2DA11CEE9D0AE7733E07830F
Call-ID: c7142b90f8c94668807a382f552a6770
```

```
CSeq: 2 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4849;ms-received-cid=900
Contact: <sip:192.0.2.1:4849;transport=tls;msreceivedcid=900>;expires=7200;+sip.instance=
"<urn:uuid:124841E4264D52E896C5D22AA8CDC316>";gruu="sip:alice@contoso.com;opaque=user:epi
d:5EFIEk0m6FKWxdIqqM3DFgAA;gruu"
Expires: 7200
Content-Length: 0
```

7. The protocol client receives the message, locates the SA, and uses it to call into the Kerberos implementation to verify the signature using its **GSS_VerifyMIC** call. The buffer that it creates for signature verification is identical to the buffer created by the server when it generated the signature.

4.3 Kerberos Authentication Example for version 4 of the protocol

1. Alice's SIP protocol client sends a REGISTER request with no **authorization** header field to the SIP server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4849
From: <sip:alice@contoso.com>;tag=22fafb15b8;epid=2ebb6f264f
To: <sip:alice@contoso.com>
Call-ID: c7142b90f8c94668807a382f552a6770
CSeq: 1 REGISTER
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841E4-
264D-52E8-96C5-D22AA8CDC316>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

2. Authentication is enabled at the server, which then challenges Alice's protocol client. The server indicates support for NTLM and Kerberos in the challenge and returns the **realm** and **targetname** values that it created during initialization, the version of the authentication protocol that it implements, and the **Date** header field.

```
SIP/2.0 401 Unauthorized
Date: Sun, 16 Dec 2007 05:11:19 GMT
WWW-
Authenticate: NTLM realm="SIP Communications Service", targetname="server.contoso.com", v
ersion=4
WWW-
Authenticate: Kerberos realm="SIP Communications Service", targetname="sip/server.contoso
.com", version=4
From: <sip:alice@contoso.com>;tag=22fafb15b8;epid=2ebb6f264f
To: <sip:alice@contoso.com>;tag=9588410E2DA11CEE9D0AE7733E07830F
Call-ID: c7142b90f8c94668807a382f552a6770
CSeq: 1 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4849;ms-received-cid=900
Content-Length: 0
```

3. The protocol client decides to use Kerberos and creates an SA with data from the **authentication** header field, specifically, **Kerberos**, **realm**, **targetname**, and **version**. It obtains a Kerberos ticket for the service principal (2) in the **targetname** parameter (sip/server.contoso.com) and calls the Kerberos authentication protocol implementation with it and the **Identify** and **Integrity** parameters to initialize the security context and generate a **KRB_AP_REQ** token. Because the server indicated support for version 4 of the protocol in the challenge and the protocol client supports version 4, the protocol client generates a **crand** parameter value and calls Kerberos **GSS_GetMic()** with the following buffer, which is based on the content of the REGISTER request.

```
<Kerberos><1d7d4ecf><SIP Communications Service><sip/server.contoso.com><c7142b90f8c94668
807a382f552a6770><2><REGISTER><alice@contoso.com><604168c9c0><alice@contoso.com><9588410E
2DA11CEE9D0AE7733E07830F><><><>
```

4. The protocol client encodes the Kerberos **KRB_AP_REQ** token as the **gssapi-data** parameter and the result of the **GSS_GetMic()** call as the **response** parameter using the base16 algorithm, replaces characters 'A' through 'F' in the output with their lowercase equivalents ('a' through 'f'), and sends the following request to the server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4849
From: <sip:alice@contoso.com>;tag=604168c9c0;epid=2ebb6f264f
To: <sip:alice@contoso.com>
Call-ID: c7142b90f8c94668807a382f552a6770
CSeq: 2 REGISTER
Authorization: Kerberos qop="auth", realm="SIP Communications Service", targetname="sip/s
erver.contoso.com", gssapi-data="1234ABCDEF", version=4, crand="1d7d4ecf", cnum="1",
response="4321abcdef"
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841
E4-264D-52E8-96C5-D22AA8CDC316>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

5. The server extracts the protocol client endpoint identity as the address-of-record in the **From** header field ("alice@contoso.com") and either the value of the **epid** parameter in the **From** header field ("2ebb6f264f") or the value of the **+sip.instance** parameter in the **Contact** header field ("urn:uuid:124841E4-264D-52E8-96C5-D22AA8CDC316"). It creates the SA for the Kerberos protocol, initializing it with a protocol client endpoint identifier, version ("4"), the generated **opaque** value ("A9A0BB9C"), and calls into the Kerberos implementation to initialize the security context and process the **KRB_AP_REQ** token. The Kerberos implementation authenticates the protocol client. The server also extracts the signature from the **response** parameter and calls Kerberos **GSS_VerifyMIC** to verify it against the buffer, which is the same as constructed by the protocol client in step 3. The server then extracts the user identity from the authentication protocol context and validates that the user is authorized to use the "alice@contoso.com" address-of-record. The server can now continue processing the REGISTER request, as described in [RFC3261], and pass it to the SIP registrar component.

6. After the SIP registrar component completes processing, it sends back a 200 OK response to the protocol client, which is processed by the authentication component on the server. The component locates the SA based on the reference it stored in the **Via** header field, or on some other mechanism, and it calls into the Kerberos authentication protocol implementation to generate a signature token for the response. Based on the content of the response, the server constructs the following buffer for the **GSS_GetMic()** call to Kerberos.

```
<Kerberos><211639C4><SIP Communications Service><sip/server.contoso.com><c7142b90f8c94668
807a382f552a6770><2><REGISTER><alice@contoso.com><604168c9c0><alice@contoso.com><><><><><72
00>
```

7. Kerberos returns the signature and the server creates an **authentication information** header field and sends the following message to the protocol client.

```
SIP/2.0 200 OK
AuthenticationInfo: Kerberos rspauth="602306092", srand="211639C4", snum="1", opaque="A9A
0BB9C", qop="auth", targetname="sip/server.contoso.com", realm="SIP Communications Servic
e"
From: <sip:alice@contoso.com>;tag=604168c9c0;epid=2ebb6f264f
To: <sip:alice@contoso.com>;tag=9588410E2DA11CEE9D0AE7733E07830F
```

```
Call-ID: c7142b90f8c94668807a382f552a6770
CSeq: 2 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4849;ms-received-cid=900
Contact: <sip:192.0.2.1:4849;transport=tls;msreceivedcid=900>;expires=7200;+sip.instance=
"<urn:uuid:124841E4264D52E896C5D22AA8CDC316>";gruu="sip:alice@contoso.com;opaque=user:epi
d:5EFIEk0m6FKWxdIqqM3DFgAA;gruu"
Expires: 7200
Content-Length: 0
```

8. The protocol client receives the message, locates the SA, and uses it to call into the Kerberos implementation to verify the signature using its **GSS_VerifyMIC** call. The buffer that it creates for signature verification is identical to the buffer created by the server when it generated the signature.

4.4 TLS-DSK Authentication Example for version 4 of the protocol

1. Alice's SIP protocol client sends a REGISTER request with no **authorization** header field to the SIP server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4849
From: <sip:alice@contoso.com>;tag=22fafb15b8;epid=2ebb6f264f
To: <sip:alice@contoso.com>
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 1 REGISTER
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841E4-
264D-52E8-96C5-D22AA8CDC316>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

2. Authentication is enabled at the server, which then challenges Alice's client. The server indicates support for TLS-DSK, Kerberos, and NTLM in the challenge and returns the **realm** and **targetname** values that it created during initialization, the version of the authentication protocol that it implements, and the **Date** header field.

```
SIP/2.0 401 Unauthorized
Date: Sun, 16 Dec 2007 05:11:19 GMT
WWW-Authenticate: TLS-
DSK realm="SIP Communications Service", targetname="server.contoso.com", version=4
WWW-
Authenticate: Kerberos realm="SIP Communications Service", targetname="sip/server.contoso.com
", version=4
WWW-
Authenticate: NTLM realm="SIP Communications Service", targetname="server.contoso.com", versi
on=4
From: <sip:alice@contoso.com>;tag=22fafb15b8;epid=2ebb6f264f
To: <sip:alice@contoso.com>;tag=9588410E2DA11CEE9D0AE7733E07830F
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 1 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4849;ms-received-cid=900
Content-Length: 0
```

3. The client decides to use TLS-DSK and creates an SA with data from the **authentication** header field, specifically **TLS-DSK**, **realm**, **targetname**, and **version**. It uses the TLS protocol implementation to generate a **client_hello** handshake message, which the client then encodes as the **gssapi-data** parameter, using the base64 algorithm, and sends the following request to the server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4849
From: <sip:alice@contoso.com>;tag=604168c9c0;epid=2ebb6f264f
To: <sip:alice@contoso.com>
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 2 REGISTER
Authorization: TLS-
DSK qop="auth", realm="SIP Communications Service", targetname="server.contoso.com", gssa
pi-data="FgMBAJMBAACPAwFJ5nVu5crf6v0bJApguL4gJbjafFaRyH7qNr", version=4
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841
E4-264D-52E8-96C5-D22AA8CDC316>"
User Agent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

4. The server extracts the client endpoint identity as the address-of-record in the **From** header field ("alice@contoso.com") and either the value of the **epid** parameter in the **From** header field ("2ebb6f264f") or the value of the **+sip.instance** parameter in the **Contact** header field ("urn:uuid:124841E4264D52E896C5D22AA8CDC316"). It creates the SA for the TLS-DSK protocol, initializing it with the client endpoint identifier, version ("3"), generated **opaque** value ("72118CF0"), and calls into the TLS implementation to process a **client_hello** message. The TLS implementation creates the negotiation context and generates a response that encapsulates several TLS records containing a TLS **server_hello** handshake message, followed by a TLS certificate, such as **server_key_exchange**, and then **certificate_request** and **server_hello_done** handshake messages. The server encodes the TLS response as the **gssapi-data** parameter, using the base64 algorithm, and sends it back to the client in the following 401 Unauthorized response.

```
SIP/2.0 401 Unauthorized
WWW-Authenticate: TLS-DSK opaque="72118CF0", gssapi-data="FgMBCJasdasd",
targetname="server.contoso.com", realm="SIP Communications Service", version=4
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>;tag=0858513FA91D3AAE1A5840DDB99599DF
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 2 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4320;ms-received-cid=1C500
Content-Length: 0
```

5. The client locates the SA that it created for the first challenge message from the server, decodes the value of the **gssapi-data** parameter using the base64 algorithm, and passes it along with the security context information stored in the SA down to the TLS implementation. The client obtains or locates a previously obtained certificate, and calls the TLS implementation to generate an output token that carries TLS **certificate**, **client_key_exchange**, **certificate_verify**, **change_cipher_spec**, and **finished** handshake messages. The client then encodes the TLS token as the **gssapi-data** parameter, using the base64 algorithm, generates the **authorization** header field, and sends the following request to the server.

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4320
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=2ebb6f264f
To: <sip:alice@contoso.com>
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 3 REGISTER
Authorization: TLS-
DSK opaque="72118CF0", qop="auth", realm="SIP Communications Service", targetname="server.con
toso.com", gssapi-data="FgMBAzasdasd", version=4
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841E4-
264D-52E8-96C5-D22AA8CDC316>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
Content-Length: 0
```

6. The server extracts the client endpoint identity from the request and the **opaque** value from the **authorization** header field, finds the existing SA, decodes the value of the **gssapi-data** parameter using the base64 algorithm, and passes it, along with the security context information stored in the SA, down to the TLS implementation. The TLS implementation validates the information passed to it and generates a response that encapsulates one or more TLS records that contain TLS **change_cipher_spec** and **finished** handshake messages. At this point, the server notes that TLS negotiation resulted in selecting a ciphersuite with **SHA-1 hash** function. The server also computes, or derives, client and server authentication keys, as described in section 3.3.5.2. The server then encodes the TLS response as the **gssapi-data** parameter, using the base64 algorithm, and sends it back to the client in the following 401 Unauthorized response.

```
SIP/2.0 401 Unauthorized
WWW-Authenticate: TLS-DSK opaque="72118CF0", gssapi-
data="FAMBAAEBFgasdasd", targetname="server.contoso.com", realm="SIP Communications Service",
 version=4
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>;tag=0858513FA91D3AAE1A5840DDB99599DF
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 3 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4320;ms-received-cid=1C500
Content-Length: 0
```

7. The client locates the SA that it created for the first challenge message from the server and used in the second challenge, decodes the value of the **gssapi-data** parameter using the base64 algorithm, and passes it, along with the security context information stored in the SA, down to the TLS implementation for validation. Once server information is validated, the client notes that the TLS negotiation resulted in selecting a ciphersuite with SHA-1 hash function. The client also computes, or derives, client and server authentication keys as described in section 3.2.5.1. Because the server indicated support for version 4 of the protocol in the challenge, and the client also supports version 4, the client generates a **crand** parameter value, and performs the HMAC computation with the client authentication key and the following buffer, which is based on the content of the REGISTER request.

```
<TLS-
DSK><1d7d4ecf><1><SIP Communications Service><server.contoso.com><d5f2b95d5be64c2cbfb38aa5d3a
87ae7><4><REGISTER><alice@contoso.com><4a2b44d131><alice@contoso.com><><><><>
```

8. The client encodes the result of the HMAC computation call as the **response** parameter, using the base16 algorithm, replaces characters 'A' through 'F' in the output with their lowercase equivalents ('a' through 'f'), and sends the following request to the server (2).

```
REGISTER sip:contoso.com SIP/2.0
Via: SIP/2.0/TLS 192.0.2.1:4849
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 4 REGISTER
Authorization: TLS-
DSK qop="auth", realm="SIP Communications Service", targetname="server.contoso.com", version=
4, crand="1d7d4ecf", cnum="1", response="4321abcdef"
Contact: <sip:192.0.2.1:4849;transport=tls>;proxy=replace;+sip.instance="<urn:uuid:124841E4-
264D-52E8-96C5-D22AA8CDC316>"
UserAgent: UCCP/2.0.6362.0 OC/2.0.6362.0 (Microsoft Office Communicator)
Supported: gruu-10
```

```
Content-Length: 0
```

9. The server extracts the client endpoint identity from the request and the **opaque** value from the **authorization** header field, finds the existing SA, decodes the value of the **response** parameter, using the base16 algorithm, and performs HMAC with the client authentication key and the buffer, which is the same as constructed by the client in step 7. The server then extracts the user identity from the authentication protocol context and validates that the user is authorized to use the "alice@contoso.com" address-of-record. The server can now continue processing the REGISTER request, as described in [RFC3261], and pass it to the SIP registrar component.

10. After the registrar component completes processing, it sends back a 200 OK response to the client, which is processed by the authentication component on the server. The component locates the SA based on the reference it stored in the **Via** header field, or on some other mechanism, and it performs the HMAC computation using the server authentication key and the following buffer, constructed based on the following 200 OK response.

```
<TLS-
DSK><211639C4><SIP Communications Service><server.contoso.com><d5f2b95d5be64c2cbfb38aa5d3a87a
e7><4><REGISTER><alice@contoso.com><4a2b44d131><alice@contoso.com><9588410E2DA11CEE9D0AE7733E
07830F><><><7200>
```

11. The server creates an **authentication information** header field with the result of the preceding HMAC computation, and sends the following response to the client.

```
SIP/2.0 200 OK
AuthenticationInfo: TLS-
DSK rspauth="602306092", srand="211639C4", snum="1", opaque="A9A0BB9C", qop="auth", targetnam
e="sip/server.contoso.com", realm="SIP Communications Service"
From: <sip:alice@contoso.com>;tag=4a2b44d131;epid=8248ca9ebb
To: <sip:alice@contoso.com>;tag=9588410E2DA11CEE9D0AE7733E07830F
Call-ID: d5f2b95d5be64c2cbfb38aa5d3a87ae7
CSeq: 4 REGISTER
Via: SIP/2.0/TLS 192.0.2.1:4849;ms-received-cid=900
Contact: <sip:192.0.2.1:4849;transport=tls;msreceivedcid=900>;expires=7200;+sip.instance="<ur
n:uuid:124841E4264D52E896C5D22AA8CDC316>";gruu="sip:alice@contoso.com;opaque=user:epid:5EFIEk
0m6FKWxdIqqM3DFgAA;gruu"
Expires: 7200
Content-Length: 0
```

12. The client receives the message, locates the SA, and uses it to compute the HMAC over the server (2) authentication (2) key and buffer created based on the message data. The buffer that it creates for signature verification is identical to the buffer created by the server (2) when it generated the signature.

4.5 Digest Authentication Example for Anonymous Join

1. Alice sends an anonymous INVITE without any **authorization** header field to the conference **focus**.

```
INVITE sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R7G SIP/2.0
Via: SIP/2.0/TLS 157.56.64.61:13184
Max-Forwards: 70
From: "Alice" <sip:6551156d569c4b7d945f310ff10943c5@anonymous.invalid>;tag=c9ef6b0990;epid=c3
2b51b28c
To: sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R7G
Call-ID: 6d5b48eabee745c49dcf7e064c37cbe9
```

```
CSeq: 1 INVITE
Contact: <sip:6551156d569c4b7d945f310ff10943c5@anonymous.invalid>;proxy=replace;+sip.instance
="<urn:uuid:782873E3-EC25-5E64-B374-0FF05E0839A5>"
```

2. Authentication is enabled at the server, which then challenges Alice's client. The server indicates support for digest in the challenge and returns the **realm** value that it created during initialization and the version of the authentication protocol that it implements.

```
SIP/2.0 401 Unauthorized
Date: Thu, 25 Feb 2010 22:53:49 GMT
WWW-Authenticate: Digest realm="bob@contoso.com", nonce="h8A4ZW22ygGZozIIGZcb43waVME-M6Gq",
opaque="0C1D4536", algorithm=MD5-sess, qop="auth"
From: "Alice" <sip:6551156d569c4b7d945f310ff10943c5@anonymous.invalid>;tag=c9ef6b0990;epid=c3
2b51b28c
To:<sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R7G>;tag=B19EA55CEC3D9316761BE6483
19D2FA0
Call-ID: 6d5b48eabee745c49dcf7e064c37cbe9
CSeq: 1 INVITE
Via: SIP/2.0/TLS 157.56.64.61:13184;received=157.54.78.7;ms-received-port=13184;ms-received-
cid=63995800
Content-Length: 0
```

3. The client creates an SA with data from the **authentication** header field, specifically, **Digest**, **realm**, and **version**. It hashes the user credential using the requested algorithm with the **nonce**, **nonce-count**, and **cnonce** values. The client then sends the **digest** in the response parameter of the **authorization** header.

```
INVITE sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R7G SIP/2.0
Via: SIP/2.0/TLS 157.56.64.61:13184
Max-Forwards: 70
From: "Alice" <sip:6551156d569c4b7d945f310ff10943c5@anonymous.invalid>;tag=c9ef6b0990;epid=c3
2b51b28c
To: sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R7G
Call-ID: 6d5b48eabee745c49dcf7e064c37cbe9
CSeq: 2 INVITE
Contact: <sip:6551156d569c4b7d945f310ff10943c5@anonymous.invalid>;proxy=replace;+sip.instance
="<urn:uuid:782873E3-EC25-5E64-B374-0FF05E0839A5>"
Authorization: Digest username="6551156d-569c-4b7d-945f-310ff10943c5", realm="bob@contoso.com
", qop=auth, algorithm=MD5sess, uri="sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R
7G", nonce="h8A4ZW22ygGZozIIGZcb43waVMEM6Gq", nc=1, cnonce="", opaque="0C1D4536",
response="b4543cd4d6a923b4ab4fd4583af48f0e"
```

4. The server validates the conference PIN by verifying the digest that was passed in the response parameter of the **authorization** header field and returns a success response back to the client. Alice has successfully joined the conference.

```
SIP/2.0 200 Invite dialog created
From: "Alice" <sip:6551156d569c4b7d945f310ff10943c5@anonymous.invalid>;tag=c9ef6b0990;epid=c3
2b51b28c
To: <sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R7G>;tag=80730080
Call-ID: 6d5b48eabee745c49dcf7e064c37cbe9
CSeq: 2 INVITE
Via: SIP/2.0/TLS 157.56.64.61:13184;received=157.54.78.7;ms-received-port=13184;ms-received-
cid=63995800
Contact: <sip:bob@contoso.com;gruu;opaque=app:conf:focus:id:854T0R7G>;isfocus
```

5 Security

5.1 Security Considerations for Implementers

The proprietary extensions defined in this document do not require any special security considerations beyond what is natively defined for SIP, NTLM, and Kerberos protocols.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Microsoft Office Communications Server 2007
- Microsoft Office Communicator 2007
- Microsoft Office Communications Server 2007 R2
- Microsoft Office Communicator 2007 R2
- Microsoft Lync Server 2010
- Microsoft Lync 2010
- Microsoft Lync Server 2013
- Microsoft Lync Client 2013/Skype for Business
- Microsoft Skype for Business 2016
- Microsoft Skype for Business Server 2015

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 3.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<2> Section 3.2.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<3> Section 3.2.2: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported

<4> Section 3.2.4.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<5> Section 3.2.4.3: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: Anonymous Client sessions are not supported.

<6> Section 3.2.4.5: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<7> Section 3.2.5.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: Anonymous Client sessions are not supported.

<8> Section 3.2.5.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<9> Section 3.2.5.2: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<10> Section 3.2.5.4: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<11> Section 3.2.5.5: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: Anonymous Client sessions are not supported.

<12> Section 3.2.5.6: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: Anonymous Client sessions are not supported.

<13> Section 3.3.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<14> Section 3.3.1: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<15> Section 3.3.1: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<16> Section 3.3.2: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<17> Section 3.3.3: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<18> Section 3.3.4.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<19> Section 3.3.5.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<20> Section 3.3.5.1: Office Communications Server 2007 "Communications Server 2007 Update Package: March 2008" (http://support.microsoft.com/?kbid=949260) not installed implements version 3 of the authentication protocol.

<21> Section 3.3.5.1: Office Communications Server 2007 with "Communications Server 2007 Update Package: March 2008" (http://support.microsoft.com/?kbid=949260) or later update package installed implements version 4 of the authentication protocol

<22> Section 3.3.5.1: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: Anonymous Client sessions are not supported.

<23> Section 3.3.5.2: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<24> Section 3.3.5.3: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: TLS-DSK authentication protocol is not supported.

<25> Section 3.3.5.4: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: Anonymous Client sessions are not supported.

<26> Section 3.3.5.5: Office Communications Server 2007, Office Communicator 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2: Anonymous Client sessions are not supported.

<27> Section 3.3.5.7: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index

N

O

P

R

S

T

V

W