[MS-XOAUTH]:

OAuth 2.0 Authorization Protocol 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.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **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.

Support. For questions and support, please contact dochelp@microsoft.com.

Preliminary Documentation. This particular Open Specifications document provides documentation for past and current releases and/or for the pre-release version of this technology. This document provides final documentation for past and current releases and preliminary documentation, as applicable and specifically noted in this document, for the pre-release version. Microsoft will release final documentation in connection with the commercial release of the updated or new version of this technology. Because this documentation might change between the pre-release version and the final

version of this technology, there are risks in relying on this preliminary documentation. To the extent that you incur additional development obligations or any other costs as a result of relying on this preliminary documentation, you do so at your own risk.

Revision Summary

Date	Revision History	Revision Class	Comments
4/27/2012	0.1	New	Released new document.
7/16/2012	0.1	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	1.0	Major	Significantly changed the technical content.
2/11/2013	2.0	Major	Significantly changed the technical content.
7/26/2013	2.1	Minor	Clarified the meaning of the technical content.
11/18/2013	2.1	None	No changes to the meaning, language, or formatting of the technical content.
2/10/2014	2.1	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	2.1	None	No changes to the meaning, language, or formatting of the technical content.
7/31/2014	2.1	None	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	2.1	None	No changes to the meaning, language, or formatting of the technical content.
5/26/2015	3.0	Major	Significantly changed the technical content.
9/14/2015	3.1	Minor	Clarified the meaning of the technical content.
6/13/2016	3.2	Minor	Clarified the meaning of the technical content.
9/14/2016	3.2	None	No changes to the meaning, language, or formatting of the technical content.
7/24/2018	4.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

The OAuth 2.0 Authorization Protocol Extensions extend the OAuth 2.0 Authentication Protocol and the JSON Web Token (JWT) to enable server-to-server authentication.

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:

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].

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

Hypertext Transfer Protocol Secure (HTTPS): An extension of HTTP that securely encrypts and decrypts web page requests. In some older protocols, "Hypertext Transfer Protocol over Secure Sockets Layer" is still used (Secure Sockets Layer has been deprecated). For more information, see [SSL3] and [RFC5246].

private key: One of a pair of keys used in public-key cryptography. The private key is kept secret and is used to decrypt data that has been encrypted with the corresponding public key. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

public key: One of a pair of keys used in public-key cryptography. The public key is distributed freely and published as part of a digital certificate. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

realm: An administrative boundary that uses one set of authentication servers to manage and deploy a single set of unique identifiers. A realm is a unique logon space.

realm autodiscovery: A process used by client applications to obtain the name of a server resource's source **realm** and then use that information to locate a **security token service (STS)** that can issue access tokens to the resource.

security principal: A unique entity that is identifiable through cryptographic means by at least one key. It frequently corresponds to a human user, but also can be a service that offers a resource to other security principals. Also referred to as principal.

security principal identifier: A value that is used to uniquely identify a security principal. In Windows-based systems, it is a security identifier (SID). In other types of systems, it can be a user identifier or other type of information that is associated with a security principal.

security token: An opaque message or data packet produced by a Generic Security Services (GSS)-style authentication package and carried by the application protocol. The application has no visibility into the contents of the token.

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

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. TCP handles keeping

track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

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 principal name (UPN): A user account name (sometimes referred to as the user logon name) and a domain name that identifies the domain in which the user account is located. This is the standard usage for logging on to a Windows domain. The format is: someone@example.com (in the form of an email address). In Active Directory, the userPrincipalName attribute of the account object, as described in [MS-ADTS].

X.509: An ITU-T standard for public key infrastructure subsequently adapted by the IETF, as specified in [RFC3280].

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.

[IETFDRAFT-JWT-LATEST] Jones, M., Bradley, J., and Sakimura, N., "JSON Web Token (JWT) draft-ietf-oauth-json-web-token-08", draft-ietf-oauth-json-web-token-08, May 2013, http://datatracker.ietf.org/doc/draft-ietf-oauth-json-web-token/

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-OAUTH2EX] Microsoft Corporation, "OAuth 2.0 Authentication Protocol Extensions".

[MS-ODATA] Microsoft Corporation, "Open Data Protocol (OData)".

[MS-SPSTWS] Microsoft Corporation, "SharePoint Security Token Service Web Service Protocol".

[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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., et al., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.rfc-editor.org/rfc/rfc2617.txt

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

[RFC793] Postel, J., Ed., "Transmission Control Protocol: DARPA Internet Program Protocol Specification", RFC 793, September 1981, http://www.rfc-editor.org/rfc/rfc793.txt

1.2.2 Informative References

[MS-SPS2SAUTH] Microsoft Corporation, "OAuth 2.0 Authentication Protocol: SharePoint Profile".

1.3 Overview

These extensions specify how applications can perform server-to-server authentication using a **security token service (STS)**. For example, an email service might use these extensions to authenticate itself when it makes a call to an instant messaging service. Both of these services are server applications. However, in the scope of this protocol, the email service would be the client, and the instant messaging service would be the server. For an example of a server-to-server **security token** that a client might send to authenticate itself, see section 4.2.

1.4 Relationship to Other Protocols

These extensions extend the OAuth 2.0 Authentication Protocol, as described in [MS-OAUTH2EX], and JSON Web Token (JWT), as described in [IETFDRAFT-JWT-LATEST].

For information on how to implement an **STS**, see [MS-SPSTWS].

For conceptual background information and overviews of the relationships and interactions between this and other protocols, see [MS-OXPROTO].

1.5 Prerequisites/Preconditions

The client is required to reside in the same **realm** as the **STS** to request a server-to-server **security token** from it.

1.6 Applicability Statement

These extensions apply only when a service call is made to or from an application that supports server-to-server authentication.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

These extensions transport messages over **TCP**, as specified in [RFC793], and do not pass any specific parameters to the transport. Transport-layer security MUST be used to secure the **security tokens**. These extensions use **HTTPS**, as specified in [RFC2818], to do so.

Messages sent using these extensions are not encoded by Open-Data, as specified in [MS-ODATA], and use the default character set defined by the client or the server.

Security tokens are JWTs and are sent using **HTTP Authorization** headers, as specified in [IETFDRAFT-JWT-LATEST]. **HTTP Authorization** headers are specified in [RFC2617].

2.2 Message Syntax

A **security principal** is represented as a **security principal identifier** in the messages sent by applications. A security principal identifier is a **GUID**.

For more details about the messages typically exchanged between a client and an **STS**, see [MS-SPSTWS].

For clarity, this document uses different names to refer to the server-to-server **security tokens** that are exchanged in various scenarios. An actor token is a signed security token that is issued by an STS, or by the client itself if the server trusts it to do so. An outer token is an unsigned security token that is constructed by the client and contains user information in addition to an actor token. In this scenario, the actor token is referred to as the inner token. All of these security tokens are formatted in the same way, as specified in [IETFDRAFT-JWT-LATEST], and contain the claims and header fields specified in this section.

The following table describes claims that are exchanged in server-to-server security tokens. The claim values are all of data type **STRING**, as specified in [MS-DTYP].

Claim type	Claim value description	Example claim values
aud	The targeted service for which the client issued the server-to-server security token.	<security principal identifier>/<hostname>@<realm>
iss	The security principal identifier of the server-to-server security token issuer.	<security principal identifier>@<realm>
nameid	The logged on user's **user principal name (UPN)** value for the security principal that made the request.	user@contoso.com
nbf	The time at which the server-to-server security token was created.	129592882368666656
exp	The time at which the server-to-server security token expires.	129592882368666656
trustedfordelegation	"true" if the client is trusted to delegate a user identity; otherwise, "false".	true false
identityprovider	The identity provider that authenticated the caller.	windows forms trusted

Claim type	Claim value description	Example claim values
actort	The security token issued and signed by the STS. An actor token has the same format as any other security token.	See section 4.3 and section 4.4.
smtp	The logged on user's email address.	user@contoso.com
sip	The logged on user's sip address.	user@contoso.com
msexchuid	A unique identifier that the STS can give the user. This is an additional claim that the STS adds and is not required by the OAuth 2.0 Authentication Protocol, as specified in [MS-OAUTH2EX].	objectGUID@contoso.com
appctx	The application context. This claim contains a subset of claims that is specific to the service accessed by the client.	See section 4.5.

The following list describes the header fields in a server-to-server security token. The field values are all of data type **STRING**, as specified in [MS-DTYP].

- **typ**. The token type. The value MUST always be "JWT".
- **alg**. The algorithm used to encrypt the contents of the token. The value of this field MUST be either "none" or "rs256". Actor tokens are always signed and have **alg** fields that contain the value "rs256". Outer tokens that contain inner signed tokens, as described in section 4.3 and section 4.4, are not signed and have **alg** fields that contain the value "none".
- **x5t**. The **base64** encoded thumbprint of the certificate used to sign the security token. This field is optional.

The header fields are contained in a separate part of the security token, as specified in [IETFDRAFT-JWT-LATEST].

3 Protocol Details

3.1 Client Details

Note that the client and server are in fact both server applications. However, in the scope of this protocol, one server application acts as the client, and one server application acts as the server.

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

3.1.5.1 Realm Autodiscovery Through HTTP 401 Challenge

A client can use **realm autodiscovery** to obtain the name of its **realm**, which it then uses to locate an **STS**. It uses realm autodiscovery by sending a request to the server that contains an empty **Bearer HTTP Authorization** header to the server. The **HTTP Authorization** header is specified in [RFC2617].

For an example of realm autodiscovery through HTTP 401 challenge, see section 4.6.

3.1.5.2 Server-to-Server Security Token Contents

The server-to-server **security token** sent by the client MUST be compatible with the JWT format specified in [IETFDRAFT-JWT-LATEST] and [MS-OAUTH2EX].

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Authentication Within a Single Organization

The following procedure shows the authentication that takes place when a client makes a call to a server in the same organization using these extensions.

1. The organization's IT administrator sets up an **STS** and configures it with the **security principal identifiers** for the client and server. The client and server each exchange **public keys**, carried in **X.509** certificates, with the STS. The administrator also configures the client and server to trust **security tokens** issued by the STS.
2. The client makes an anonymous request to the server.
3. The server responds with an HTTP 401 challenge. HTTP 401 is specified in [RFC2616] and [RFC2617].
4. The client requests a security token from the STS. It does this by sending a self-issued security token that is signed with its **private key**. The security token contains the **aud**, **iss**, **nameid**, **nbf**, **exp**, and **trustedfordelegation** claims as specified in section 2.2. The client request also includes a *resource* parameter and a *realm* parameter, as specified in [MS-OAUTH2EX]. The value of the *resource* parameter is the **Uniform Resource Identifier (URI)** of the server. For an example of a self-issued security token, see section 4.2.
5. The STS validates the public key of the security token provided by the client, verifies that the client is authorized to access the requested resource, and responds to the client with a server-to-server security token that is signed with a public key that the server trusts. The security token contains the **aud**, **iss**, **nameid**, **nbf**, **exp**, and **identityprovider** claims, as specified in section 2.2. For an example of a server-to-server security token issued by an STS, see section 4.1.
6. The client sends the server-to-server security token to the server.
7. The server validates the server-to-server security token by checking the values of the **aud**, **iss**, and **exp** claims and the public key provided by the STS. It performs additional validation checks to ensure that the client is authorized to access the requested resource. It then responds to the client with the requested resource.

3.2.5.2 Authentication with User Information Within a Single Organization

The following procedure shows the authentication that takes place when a client makes a call to a server in the same organization using these extensions. The client also sends user information to the server, and the server uses the information to determine whether to return the requested resource.

1. The organization's IT administrator sets up an **STS** and configures it with the **security principal identifiers** for the client and server. The client and server each exchange **public keys**, carried in

X.509 certificates, with the STS. The administrator also configures the client and server to trust **security tokens** issued by the STS.

2. The client makes an anonymous request to the server.

3. The server responds with an HTTP 401 challenge, as specified in [RFC2616] and [RFC2617].

4. The client requests a security token from the STS. It does this by sending a self-issued security token that is signed with its **private key**. The security token contains the **aud**, **iss**, **nameid**, **trustedfordelegation**, **nbf**, and **exp** claims, as specified in section 2.2. The client request also includes a *resource* parameter and a *realm* parameter, as specified in [MS-OAUTH2EX]. The value of the *resource* parameter is the **URI** of the server. For an example of a self-issued security token, see section 4.2.

5. The STS validates the public key of the security token provided by the client, verifies that the client is authorized to access the requested resource, and responds to the client with a server-to-server security token that is signed with a public key that the server trusts. The security token contains the **aud**, **iss**, **nameid**, **nbf**, and **exp** claims. For an example of a server-to-server security token issued by an STS that contains user information, see section 4.3.

6. The client sends a self-issued security token to the server that includes the server-to-server security token it received from the STS, as well as additional claims that contain the user information. The additional claims are the **aud**, **iss**, **nameid**, **nbf**, **exp**, **smtp**, **sip**, **msexchuid**, and **actort** claims, as specified in section 2.2. The **actort** claim contains the server-to-server security token provided by the STS. Note that the server-to-server security token is signed, but the user information is not. For an example of a self-issued server-to-server security token that contains user information, see section 4.4.

7. The server validates the request by checking the user information contained in the **aud**, **iss**, and **exp** claims and the public key used to sign the security token provided by the STS. Because the user information is not signed, the server validates the user information by checking that the values of the **aud** and **iss** claims in the user information match the values of the **aud** and **iss** claims in the server-to-server security token contained in the **actort** claim. For security considerations regarding the unsigned user information, see section 5.1.

8. The server performs additional validation checks to ensure that the client is authorized to access the requested resource. It then responds to the client with the requested resource.

3.2.5.3 Authentication with Third-Party Application

The following procedure shows the authentication that takes place when a client makes a call to a third-party application in the same organization using these extensions.

1. The organization's IT administrator sets up an **STS** and configures it with the **security principal identifiers** for the client and third-party application. The client and third-party application each exchange **public keys**, carried in **X.509** certificates, with the STS. The administrator also configures the client and third-party application to trust **security tokens** issued by the STS.

2. The client makes an anonymous request to the third-party application.

3. The third-party application responds with an HTTP 401 challenge, as specified in [RFC2616] and [RFC2617].

4. The client requests a security token from the STS. It does this by sending a self-issued security token that is signed with its **private key**. The security token contains the **aud**, **iss**, **nameid**, **nbf**, and **exp** claims, as specified in section 2.2.

5. The STS validates the public key of the security token provided by the client, verifies that the client is authorized to access the requested resource, and returns a server-to-server security

token that is signed with a public key that the third-party application trusts. The security token contains the **aud**, **iss**, **nameid**, **nbf**, **exp**, and **appctx** claims, as specified in section 2.2. The **appctx** claim contains information that is implementation-specific to the third-party application. For an example of a server-to-server security token that is used to access a third-party application, see section 4.5.

6. The client sends the server-to-server security token to the third-party application.

7. The third-party application validates the server-to-server security token by checking the values of the **aud**, **iss**, and **exp** claims and the public key provided by the STS. It performs additional validation checks to ensure that the client is authorized to access the requested resource. It then responds to the client with the requested resource.

3.2.5.4 Realm Autodiscovery Through HTTP 401 Challenge

When a server receives a request that contains an empty **Bearer HTTP Authorization** header, the server responds with an HTTP 401 challenge, as specified in [RFC2616] and [RFC2617]. The **Bearer WWW-Authenticate** header of the HTTP 401 challenge contains the following fields:

- **client_id**. The client's **security principal identifier**.
- **realm**. The server MAY<1> return this field. This is the source **realm** of the client.
- **trusted_issuers**. A comma-separated list of all **security token** issuers the server trusts. The client can then select a security token issuer to request a security token.

For an example of **realm autodiscovery** through HTTP 401 challenge, see section 4.6.

3.2.5.5 Server-to-Server Security Token Contents

The server can accept server-to-server **security tokens** with the claim types specified in section 2.2.

3.2.5.6 Server-to-Server Validation Criteria

The server accepts a server-to-server **security token** that meets the following criteria:

- The server-to-server security token is signed with a trusted signing certificate from an **STS** that the server trusts.
- The server-to-server security token contains an **iss** claim whose value shows that the security token is issued by an STS that the server trusts.
- The server-to-server security token contains a **nameid** claim with the UPN value of the logged-on user.
- If the client constructs an unsigned outer security token to contain user information as well as a signed actor token (that is, an inner token), as described in section 4.3 and section 4.4, the value of the **iss** claim in the outer token matches the value of the **nameid** claim in the inner token. The server performs a case-sensitive comparison.
- The server-to-server security token contains an **aud** claim whose value meets the following criteria:
  - The **aud** claim value MUST contain three parts: client_id, hostname, and realm.
  - The value of the client_id part is the **security principal identifier** of a **security principal** that the server trusts. The server performs a case-sensitive comparison.

- The value of the hostname part is the host name of the server. The server performs a case-insensitive comparison to verify that it is the target of the request.
- The value of the realm part is the source **realm**. The server performs a case-sensitive comparison.

The STS uses the claims in the server-to-server security token to authenticate the caller.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

4.1 Security Token Issued by STS

In this example, a client attempts to access a resource on the server. The server responds with an HTTP 401 challenge that lists the **security token** issuers it trusts in the **trusted_issuers** field. An example of such a challenge is as follows.

```
HTTP/1.1 401 Unauthorized
Server: Fabrikam/7.5
request-id: 443ce338-377a-4c16-b6bc-c169a75f7b00
X-FEServer: DUXYI01CA101
WWW-Authenticate: Bearer client_id="00000002-0000-0ff1-ce00-000000000000",
trusted issuers="00000001-0001-0000-c000-000000000000@*"
WWW-Authenticate: Basic Realm=""
X-Powered-By: ASP.NET
Date: Thu, 19 Apr 2012 17:04:16 GMT
Content-Length: 0
```

1. The client sends its credentials to the indicated token issuer, which is an **STS**.
2. The STS authenticates the client and issues an actor token to the client.
3. The client uses the actor token to access the resource it requested on the server.

The following is an example of an actor token issued by an STS. For more information about the claim values contained in this security token, see section 2.2.

```
actor:
{
    "typ":"JWT",
    "alg":"RS256",
    "x5t":"XqrnFEfsS55_vMBpHvF0pTnqeaM"
}.{
    "aud":"00000003-0000-0ff1-ce00-000000000000/contoso.com@b84c5afe-7ced-4ce8-aa0b-
df0e2869d3c8",
    "iss":"00000001-0000-0000-c000-000000000000@b84c5afe-7ced-4ce8-aa0b-df0e2869d3c8",
    "nbf":"1323380070",
    "exp":"1323383670",
    "nameid":"00000002-0000-0ff1-ce00-000000000000@b84c5afe-7ced-4ce8-aa0b-df0e2869d3c8",
    "identityprovider":"00000001-0000-0000-c000-000000000000@b84c5afe-7ced-4ce8-aa0b-
df0e2869d3c8"
}
```

4.2 Security Token Self-Issued by Client

In this example, the client tries to access a resource on the server. The server responds with an HTTP 401 challenge that indicates the **security token** issuers it trusts in the **trusted_issuers** field. An example of such a challenge is as follows.

```
HTTP/1.1 401 Unauthorized
Server: Fabrikam/7.5
request-id: 443ce338-377a-4c16-b6bc-c169a75f7b00
X-FEServer: DUXYI01CA101
WWW-Authenticate: Bearer client_id="00000002-0000-0ff1-ce00-000000000000",
trusted issuers="00000001-0001-0000-c000-000000000000@*"
WWW-Authenticate: Basic Realm=""
X-Powered-By: ASP.NET
Date: Thu, 19 Apr 2012 17:04:16 GMT
```

```
Content-Length: 0
```

1. The client is one of the token issuers trusted by the server, so it creates an actor token and signs it with its credentials.
2. The client uses the actor token to access the resource it requested on the server.

The following is an example of an actor token that is self-issued by a client. For more information about the claim values contained in this security token, see section 2.2.

```
actor:
{
    "typ":"JWT",
    "alg":"RS256",
    "x5t":"mH-TTlt-HAXC9-vjKVFtX6bAsR0"
}.{
    "aud":"00000003-0000-0ff1-ce00-000000000000/contoso.com@EXHB-
88371dom.extest.contoso.com",
    "iss":"00000002-0000-0ff1-ce00-000000000000@EXHB-88371dom.extest.contoso.com",
    "nbf":"1323380605",
    "exp":"1323409405",
    "nameid":"00000002-0000-0ff1-ce00-000000000000@EXHB-88371dom.extest.contoso.com",
    "trustedfordelegation":"true"
}
```

4.3 Security Token Issued by STS with User Information Added by Client

In this example, the client tries to access a resource on the server. The server responds with an HTTP 401 challenge that lists the **security token** issuers it trusts in the **trusted_issuers** field. An example of such a challenge is as follows.

```
HTTP/1.1 401 Unauthorized
Server: Fabrikam/7.5
request-id: 443ce338-377a-4c16-b6bc-c169a75f7b00
X-FEServer: DUXYI01CA101
WWW-Authenticate: Bearer client_id="00000002-0000-0ff1-ce00-000000000000",
trusted issuers="00000001-0001-0000-c000-000000000000@*"
WWW-Authenticate: Basic Realm=""
X-Powered-By: ASP.NET
Date: Thu, 19 Apr 2012 17:04:16 GMT
Content-Length: 0
```

1. The client sends its credentials to the indicated security token issuer, which is an **STS**.
2. The STS authenticates the client and issues an actor token to the client.
3. The client constructs an unsigned outer token that contains additional user information to provide to the server. The outer token also contains the signed actor token issued by the STS.
4. The client uses the outer token to access the resource it requested on the server.

The following is an example of an outer token that is constructed by the client and contains user information, as well as an actor token issued by an STS. For more information about the claim values contained in this security token, see section 2.2.

```
{
    "typ":"JWT",
    "alg":"none"
}.{
```

```
    "aud":"00000003-0000-0ff1-ce00-000000000000/fabrikam.com@fabrikam-oauth-
929.extest.fabrikam.com",
    "iss":"00000001-0000-0ff1-ce00-000000000000@b84c5afe-7ced-4ce8-aa0b-df0e2869d3c8",
    "nbf":"1323380069",
    "exp":"1323408869",
    "nameid":00000002-0000-0ff1-ce00-000000000000@BLID-EXHB-90232dom.extest.contoso.com
    "actort":"..actor token.."
}
```

The following is an example of an actor token, as mentioned in the previous listing.

```
{
    "typ":"JWT",
    "alg":"RS256",
    "x5t":"hEAw-SXzTNaDBUwfAh2YScnBOxA"
}.{
    "aud":"00000002-0000-0ff1-ce00-000000000000/contoso.com@EXHB-
88371dom.extest.contoso.com",
    "iss":"00000003-0000-0ff1-ce00-000000000000@e54c2f60-0ad3-4ef8-8ba2-b3ae01b35494",
    "nbf":"1346674665",
    "exp":"1346804265",
    "nameid":"00000003-0000-0ff1-ce00-000000000000@e54c2f60-0ad3-4ef8-8ba2-b3ae01b35494"
}
```

4.4 Security Token Self-Issued By Client with User Information

In this example, the client tries to access a resource on the server. The server responds with an HTTP 401 challenge that indicates the **security token** issuers it trusts in the **trusted_issuers** field. An example of such a challenge is as follows.

```
HTTP/1.1 401 Unauthorized
Server: Fabrikam/7.5
request-id: 443ce338-377a-4c16-b6bc-c169a75f7b00
X-FEServer: DUXYI01CA101
WWW-Authenticate: Bearer client_id="00000002-0000-0ff1-ce00-000000000000",
trusted_issuers="00000001-0001-0000-c000-000000000000@*"
WWW-Authenticate: Basic Realm=""
X-Powered-By: ASP.NET
Date: Thu, 19 Apr 2012 17:04:16 GMT
Content-Length: 0
```

1. The client is one of the token issuers trusted by the server, so it creates an actor token and signs it with its credentials.
2. The client constructs an unsigned outer token that contains additional user information to provide to the server. The outer token also contains the signed actor token issued by the client.
3. The client uses the outer token to access the resource it requested on the server.

The following is an example of an outer token that is constructed by the client and contains user information, as well as a security token that is self-issued by the client. For more information about the claim values contained in this security token, see section 2.2.

```
{
    "typ":"JWT",
    "alg":"none"
}.{
    "aud":"00000003-0000-0ff1-ce00-000000000000/contoso.com@EXHB-
88371dom.extest.contoso.com",
```

```
    "iss":"00000002-0000-0ff1-ce00-000000000000@EXHB-
88371dom.extest.contoso.com",
    "nbf":"1323380605",
    "exp":"1323409405",
    "nameid":"ewsuser-55a83300@EXHB-88371dom.extest.contoso.com",
    "smtp":"ewsuser-55a83300@exhb-88371dom.extest.contoso.com",
    "sip":"ewsuser-55a83300@exhb-88371dom.extest.contoso.com",
    "msexchuid":"842e4c3a-0879-4973-83f9-495bb9863e18@exhb-88371dom.extest.contoso.com",
    "actort":"..actor token.."
}
```

The following is an example of an actor token, as mentioned in the previous listing.

```
{
    "typ":"JWT",
    "alg":"RS256",
    "x5t":"hEAw-SXzTNaDBUwfAh2YScnBOxA"
}.{
    "aud":"00000002-0000-0ff1-ce00-000000000000/contoso.com@EXHB-
88371dom.extest.contoso.com",
    "iss":"00000003-0000-0ff1-ce00-000000000000@e54c2f60-0ad3-4ef8-8ba2-b3ae01b35494",
    "nbf":"1346674665",
    "exp":"1346804265",
    "nameid":"00000003-0000-0ff1-ce00-000000000000@e54c2f60-0ad3-4ef8-8ba2-b3ae01b35494"
}
```

4.5 Security Token for Accessing a Third-Party Service with Extensions

In this example, the client tries to access a resource on a third-party service. The service responds with an HTTP 401 challenge that lists the **security token** issuers it trusts in the **trusted_issuers** field. An example of such a challenge is as follows.

```
HTTP/1.1 401 Unauthorized
Server: Fabrikam/7.5
request-id: 443ce338-377a-4c16-b6bc-c169a75f7b00
X-FEServer: DUXYI01CA101
WWW-Authenticate: Bearer client_id="00000002-0000-0ff1-ce00-000000000000",
trusted_issuers="00000001-0001-0000-c000-000000000000@*"
WWW-Authenticate: Basic Realm=""
X-Powered-By: ASP.NET
Date: Thu, 19 Apr 2012 17:04:16 GMT
Content-Length: 0
```

1. The client sends its credentials to the indicated security token issuer, which is an **STS**. The credentials include an **appctx** claim that contains additional user information to be provided to the service. The STS does not examine or modify the user information in the **appctx** claim.
2. The STS authenticates the client and issues an actor token to the client that includes the **appctx** claim provided by the client.
3. The client uses the actor token to access the resource it requested on the service. The actor token includes the **appctx** claim that was previously supplied by the client and contains additional user information to be provided to the service. This scenario is analogous to the use of outer and inner tokens to convey additional user information to a server beyond what is required to authenticate to an STS, as described in section 4.3 and section 4.4.

The following is an example of a security token that is used to access a third-party service and contains user information. For more information about the claim values contained in this security token, see section 2.2.

```
{
    "aud":"00000002-0000-0ff1-ce00-000000000000/exhb-42702.exhb-
42702dom.extest.contoso.com@b84c5afe-7ced-4ce8-aa0b-df0e2869d3c8",
    "iss":"00000002-0000-0ff1-ce00-000000000000@b84c5afe-7ced-4ce8-aa0b-df0e2869d3c8",
    "nbf":"1323197012",
    "exp":"1323225812",
    "nameid":"https://www.fabrikam.com/weprintem",
    "appctx":"{
        "nameid":"ewsuser-cff3d495@BLID-EXHB-42702dom.extest.contoso.com",
        "smtp":"ewsuser-cff3d495@BLID-EXHB-42702dom.extest.contoso.com",
        "msexchuid":"842e4c3a-0879-4973-83f9-495bb9863e18@exhb-88371dom.extest.contoso.com"
    }"
}
```

4.6 Realm Autodiscovery Through HTTP 401 Challenge

In this example, the client tries to access a resource on a server. It also tries to use **realm autodiscovery** by including an empty **Bearer** authorization header in its request. An example of such a request is as follows.

```
POST https://contoso.com/autodiscover/autodiscover.svc HTTP/1.1
Content-Type: text/xml; charset=utf-8
Accept: text/xml
User-Agent: Test/1.0 (ContosoServicesClient/15.00.0424.000)
client-request-id: 00000000-0000-0000-0000-000000000000
Authorization: Bearer
Host: contoso.com
Content-Length: 1368
Expect: 100-continue
```

The server responds with an HTTP 401 challenge that lists the **security token** issuers it trusts in the **trusted_issuers** field. An example of such a challenge is as follows.

```
HTTP/1.1 401 Unauthorized
Server: Fabrikam/7.5
request-id: 443ce338-377a-4c16-b6bc-c169a75f7b00
X-FEServer: XJSUI01CA101
WWW-Authenticate: Bearer client_id="00000002-0000-0ff1-ce00-000000000000",
trusted issuers="00000001-0001-0000-c000-000000000000@*"
WWW-Authenticate: Basic Realm=""
X-Powered-By: ASP.NET
Date: Thu, 19 Apr 2012 17:04:16 GMT
Content-Length: 0
```

In this example, the server determines that the value in the **trusted_issuers** field contains sufficient information for the client to locate the **STS**, so the server does not include a **realm** field.

5 Security

5.1 Security Considerations for Implementers

The security considerations described in [MS-SPS2SAUTH] apply when implementing these extensions.

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 updates to those products.

- Microsoft Exchange Server 2013
- Microsoft Exchange Server 2016
- Microsoft SharePoint Server 2013
- Microsoft SharePoint Foundation 2013
- Microsoft SharePoint Server 2016
- Microsoft Exchange Server 2019 Preview
- Microsoft SharePoint Server 2019 Preview

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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 3.2.5.4: SharePoint Server 2013, SharePoint Foundation 2013, and SharePoint Server 2016 return this parameter.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
6 Appendix A: Product Behavior	Updated list of supported products.	Major

8 Index

R

S

T

V