# [MS-XOAUTH]: OAuth 2.0 Authorization Protocol Extensions

## Revision Summary

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 04/27/2012 | 0.1 | New | Released new document. |
| 07/16/2012 | 0.1 | No change | No changes to the meaning, language, or formatting of the technical content. |
| 10/08/2012 | 1.0 | Major | Significantly changed the technical content. |
| 02/11/2013 | 2.0 | 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: SharePoint Profile and the JSON Web Token (JWT) to enable server-to-server authentication.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

## 1.1 Glossary

The following terms are defined in [MS-GLOS]:

**base64**
**GUID**
**Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS)**
**public key**
**realm**
**security token**
**Transmission Control Protocol (TCP)**
**user principal name (UPN)**
**X.509**

The following terms are defined in [MS-OXGLOS]:

**security principal**
**security principal identifier**
**security token service (STS)**
**Uniform Resource Identifier (URI)**

The following terms are specific to this document:

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

## 1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the technical documents, which are updated frequently. References to other documents include a publishing year when one is available.

### 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. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[IETFDRAFT-JWT] Goland, Y., and Jones, M., "[OAUTH-WG] JSON Web Token (JWT) Specification Draft", September 2010, http://www.ietf.org/mail-archive/web/oauth/current/msg04407.html

[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.ietf.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.ietf.org/rfc/rfc2617.txt

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

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

### 1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-OXGLOS] Microsoft Corporation, "Exchange Server Protocols Master Glossary".

[MS-OXPROTO] Microsoft Corporation, "Exchange Server Protocols System Overview".

[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 e-mail service might use these extensions to authenticate itself when it makes a call to an instant messaging service. For an example of a server-to-server **security token** that a client application 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].

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 application 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]. **HTTP Authorization** headers are specified in [RFC2617].

## 2.2 Message Syntax

A **security principal (1)** 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], 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 (1) 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<br>false |
| **identityprovider** | The identity provider that | windows |

| Claim type | Claim value description | Example claim values |
|---|---|---|
| | authenticated the caller. | forms<br>trusted |
| **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.<br>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.<br>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].

### 2.2.1 Namespaces

None.

### 2.2.2 Custom HTTP Methods

None.

### 2.2.3 Custom HTTP Headers

None.

### 2.2.4 Common URI Parameters

None.

### 2.2.5 Elements

None.

### 2.2.6 Complex Types

None.

### 2.2.7 Simple Types

None.

### 2.2.8 Attributes

None.

### 2.2.9 Groups

None.

### 2.2.10 Attribute Groups

None.

### 2.2.11 Common Data Structures

None.

## 2.3 Directory Service Schema Elements

None.

# 3 Protocol Details

## 3.1 Client Details

### 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 application 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 application that contains an empty **Bearer HTTP Authorization** header to the server application. 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 application MUST be compatible with the JWT format specified in [IETFDRAFT-JWT] 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 application makes a call to a server application in the same organization using the server-to-server authentication protocol specified in [MS-OAUTH2EX]. Note that the client and server applications are in fact both server applications, but we use the terms "client application" and "server application" here to distinguish them based on their roles with respect to each other.

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

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

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

4. The client application requests a security token from the STS. It does this by sending a self-issued security token that is signed with its **public 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, a *realm* parameter, and an optional *state* parameter. The *resource* and *realm* parameters are defined in [MS-OAUTH2EX]. The value of the *resource* parameter is the **Uniform Resource Identifier (URI)** of the server application. 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 application, verifies that the client application 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 application trusts. The security token contains the **aud**, **iss**, **nameid**, **nbf**, **exp**, and **identityprovider** claims, as specified in section 2.2. The response also includes an optional *state* parameter. For an example of a server-to-server security token issued by an STS, see section 4.1.

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

7. The server 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 application is authorized to access the requested resource. It then responds to the client application 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 application makes a call to a server application in the same organization using the server-to-server authentication protocol specified in [MS-OAUTH2EX]. The client application also sends user information to the server application, and the server application uses the information to determine whether to return the requested resource. Note that the client and server applications are in fact both server applications, but we use the terms "client application" and "server application" here to distinguish them based on their roles with respect to each other.

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

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

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

4. The client application requests a security token from the STS. It does this by sending a self-issued security token that is signed with its public 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, a *realm* parameter, and an optional *state* parameter. The *resource* and *realm* parameters are defined in [MS-OAUTH2EX]. The value of the *resource* parameter is the URI of the server application. 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 application, verifies that the client application 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 application trusts. The security token contains the **aud**, **iss**, **nameid**, **nbf**, and **exp** claims. The response also includes an optional *state* parameter. 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 application sends a self-issued security token to the server application 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 application 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 application 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 application performs additional validation checks to ensure that the client application is authorized to access the requested resource. It then responds to the client application 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 application 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 applications. The client and third-party applications each

exchange public keys, carried in X.509 certificates, with the STS. The administrator also configures the client and third-party applications to trust security tokens issued by the STS.

2. The client application 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 application requests a security token from the STS. It does this by sending a self-issued security token that is signed with its public 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 application, verifies that the client application 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 application 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 application is authorized to access the requested resource. It then responds to the client application 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 application 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 application's security principal identifier.
- **realm**. The server application MAY<1> return this field. This is the source realm of the client application.
- **trusted_issuers**. A comma-separated list of all security token issuers the server application 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 application 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 application 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 application 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 application 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 application 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 (1) that the server application trusts. The server application performs a case-sensitive comparison.
  - The value of the hostname part is the host name of the server application. The server application 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 application 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 Application

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 application. 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 Application 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 application 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 released service packs:

- Microsoft Exchange Server 2013
- Microsoft SharePoint Server 2013
- Microsoft SharePoint Foundation 2013

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 3.2.5.4: SharePoint Server 2013 and SharePoint Foundation 2013 return this parameter.

# 7 Change Tracking

This section identifies changes that were made to the [MS-XOAUTH] protocol document between the October 2012 and February 2013 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

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 or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

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 **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

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

| Section | Tracking number (if applicable) and description | Major change (Y or N) | Change type |
|---|---|---|---|
| 1.2.1 Normative References | Removed the references [XMLNS], [XMLSCHEMA1], and [XMLSCHEMA2]. | Y | Content updated. |
| 1.2.1 Normative References | Added the reference [MS-OAUTH2EX]. | Y | Content updated. |
| 1.2.1 Normative References | Moved the reference [MS-SPS2SAUTH] to Informative References. | Y | Content updated. |
| 1.2.2 Informative References | Moved the reference [MS-SPS2SAUTH] from Normative References. | Y | Content updated. |
| 1.4 Relationship to Other Protocols | Changed reference from [MS-SPS2SAUTH] to [MS-OAUTH2EX]. | N | Content updated. |
| 2.2 Message Syntax | Moved claim descriptions from the Abstract Data Model section of the Client Details section. | N | Content updated for template compliance. |
| 2.2 Message Syntax | Changed reference from [MS-SPS2SAUTH] to [MS-OAUTH2EX]. | Y | Content updated. |
| 3.1 Client Details | Changed section title from Common Details to Client Details. | N | Content updated for template compliance. |
| 3.1.1 Abstract Data Model | Moved claim descriptions to the Message Syntax section. | N | Content updated for template compliance. |

| Section | Tracking number (if applicable) and description | Major change (Y or N) | Change type |
|---|---|---|---|
| 3.1.5.1 Realm Autodiscovery Through HTTP 401 Challenge | Added section for information moved from the Realm Autodiscovery Through HTTP 401 section Challenge in the Server Details section about how the client uses realm autodiscovery. | N | Content updated for template compliance. |
| 3.1.5.1 Realm Autodiscovery Through HTTP 401 Challenge | Clarified the usage of realm autodiscovery. | N | Content updated. |
| 3.1.5.2 Server-to-Server Security Token Contents | Added section for the information moved from the Server-to-Server Security Token Contents section in the Server Details section about the server-to-server security token that the client sends. | N | Content updated for template compliance. |
| 3.1.5.2 Server-to-Server Security Token Contents | Changed reference for server-to-server security token format from [MS-SPS2SAUTH] to [MS-OAUTH2EX]. | Y | Content updated. |
| 3.2.5.1 Authentication Within a Single Organization | Added reference for definitions of resource and realm parameters. | Y | New content added. |
| 3.2.5.1 Authentication Within a Single Organization | Changed reference for server-to-server authentication protocol from [MS-SPS2SAUTH] to [MS-OAUTH2EX]. | Y | Content updated. |
| 3.2.5.2 Authentication with User Information Within a Single Organization | Added reference for definition of resource and realm parameters. | Y | New content added. |
| 3.2.5.2 Authentication with User Information Within a Single Organization | Changed reference for server-to-server authentication protocol from [MS-SPS2SAUTH] to [MS-OAUTH2EX]. | Y | Content updated. |
| 3.2.5.4 Realm Autodiscovery Through HTTP 401 Challenge | Moved specification of how client uses realm autodiscovery to the Realm Autodiscovery Through HTTP 401 Challenge section of the Client Details section. | N | Content updated for template compliance. |
| 3.2.5.5 Server-to-Server Security Token Contents | Moved specification of the server-to-server security token that the client sends to the Server-to-Server Security Token Contents section of the Client Details section. | N | Content updated for template compliance. |

# 8 Index