Calling Xbox Live Services from Your Title Service

By: Ferdinand Schober, Xbox Advanced Technology Group, and Gregoire Guetat, Xbox Live Services

Published: July 31, 2014

Updated: May 24, 2017

**This document is confidential** and provided to you under a Microsoft non-disclosure agreement. While we have tried to ensure the accuracy of this document, we provide no express warranties or guarantees regarding the information. The information is subject to change. Microsoft may have intellectual property rights in the subject matter of this paper. This document doesn’t grant you a license to those rights—it’s for informational purposes only.

Abstract

This white paper describes how services external to Microsoft data centers can authenticate with Xbox Live security services and be authorized to access Xbox Live Services. It also covers mechanisms through which such external services can call Xbox Live Services, including Xbox Multiplayer services, on behalf of a user.

Updates to this white paper include the following (the most recent changes are highlighted in yellow for your convenience):

* June 28, 2015: Added further details for the calling patterns to the Xbox Live Multiplayer services under “user header” in the “Xbox Multiplayer Session Directory headers” section.
* September 2, 2015: Added new section “Business Partner Certificate expiration.”
* November 24, 2015: Updated Relying Party URIs to include trailing “/”.
* January 4, 2016: Removed erroneous spaces in the example Authentication header tokens in the [Calling Xbox Live Services](#_Calling_Xbox_Live) section. Added some clarification on how to add the DelegationToken claim to your services’ XSTS tokens.

Contents

[Introduction 3](#_Toc483377253)

[Configuration of Xbox Live 3](#_Toc483377254)

[Registering your service with Xbox Live 3](#_Toc483377255)

[Business Partner Certificate 3](#_Toc483377256)

[Configuring Xbox Live Services access with Business Partner Certificates 7](#_Toc483377257)

[Business Partner Certificates and sandboxes 9](#_Toc483377258)

[Additional configuration to call Xbox Live on behalf of a user: delegation 10](#_Toc483377259)

[Business Partner Certificate expiration 10](#_Toc483377260)

[Service authentication flow: overview 11](#_Toc483377261)

[Proof keys 12](#_Toc483377262)

[Obtaining a service token 14](#_Toc483377263)

[Request 14](#_Toc483377264)

[Response 17](#_Toc483377265)

[Error handling 17](#_Toc483377266)

[Obtaining an X token 18](#_Toc483377267)

[Request 18](#_Toc483377268)

[The RelyingParty property 19](#_Toc483377269)

[The PropertyBag property 20](#_Toc483377270)

[Headers 21](#_Toc483377271)

[Response 21](#_Toc483377272)

[Error handling 23](#_Toc483377273)

[Calling Xbox Live Services 25](#_Toc483377274)

[Calling Xbox Live Multiplayer services 26](#_Toc483377275)

[Xbox Multiplayer Session Directory headers 27](#_Toc483377276)

[Xbox SmartMatch headers 29](#_Toc483377277)

[References 29](#_Toc483377278)

# Introduction

With the correct setup and flow, Xbox Live Services can also be accessed from title services through service-to-service (S2S) access, which enables new gameplay and interaction scenarios. For example, custom matchmaking services can use S2S access to create and read multiplayer session information and player data.

While the RESTful request pattern for Xbox Live service endpoints is identical between the Xbox One console and title services, the authentication path is significantly different. Title services require additional authentication steps before Xbox Live service calls can be performed. The following sections describe the flow for this authentication path to enable S2S access for your service.

# Configuration of Xbox Live

Service-to-service access is not enabled by default for each title and first needs to be configured on a publisher level.

## Registering your service with Xbox Live

Before your title service can be authenticated for access to Xbox Live Services, it must first have credentials registered on Xbox Live. Specifically, your service must have a *Business Partner Certificate*, which is an X509 certificate issued by an Xbox Live certificate authority (CA). You can obtain a Business Partner Certificate through the Xbox Developer Portal (XDP) at <https://xdp.xboxlive.com>. For access to XDP, contact your Developer Account Manager.

### Business Partner Certificate

As previously noted, Business Partner Certificates are X509 certificates issued by an Xbox Live CA. In order to maintain the security of the private key to be associated with a specific certificate, the process for generating the certificate involves three steps:

1. The first step consists of generating a key container together with a public/private key pair on a computer. This can be performed on any computer running Windows; it does not need to be the server hosting your service. Use the following PowerShell script (to be run from an elevated PowerShell command prompt) to generate the key container and key pair. You will have to run this PowerShell script for each Business Partner Certificate that you want to create.

# Generate a properly sized key and make a certificate request

$certRequest = new-object -ComObject X509Enrollment.CX509CertificateRequestCertificate

$certRequest.Initialize(2) # Initialize in the machine context

$certRequest.PrivateKey.Length = 2048

$certRequest.PrivateKey.ProviderName = "Microsoft Enhanced RSA and AES Cryptographic Provider"

# Set XCN\_NCRYPT\_ALLOW\_EXPORT\_FLAG

$certRequest.PrivateKey.ExportPolicy = 2

# Subject is required by the tool even though it is overwritten when we generate the certificate

$subject = new-object -ComObject X509Enrollment.CX500DistinguishedName

$subject.Encode("CN=NOT USED")

$certRequest.Subject = $subject

# Stores private key to allow auto binding. Import the .cer file you get back into the Local Machine/Personal store to bind to the private key

$enroll = new-object -ComObject X509Enrollment.CX509Enrollment

$enroll.InitializeFromRequest($certRequest)

$strCert = $enroll.CreateRequest(0)

# Then export public key to be sent by email

$certRequest.PrivateKey.Export("PUBLICBLOB", 0x40000001)

This information is also available on XDP.

1. The output from the script run in Step 1 is the public key (that is, the CSP blob), which needs to be pasted into the Business Partner Certificate configuration section on XDP:
	1. Navigate to “Manage/Web Services” from the XDP Dashboard menu:



* 1. Create a new web service by selecting “New” and specify a name that allows you to uniquely identify the certificate:

	
	2. Select the **Certificate** icon for the new web service to open the Business Partner Certificate configuration:

	
	3. Paste the output of the script into the new certificate dialog (removing all carriage returns). If you are generating a certificate that will be used for only one sandbox, specify that sandbox in the drop-down menu. However, we recommend that you not specify a sandbox here, so that the same certificate will work across all of your development sandboxes as well as Microsoft-managed sandboxes like CERT, CERT.DEBUG, RETAIL, and so on.

	

Or (recommended):



* 1. XDP will then create and allow the download of a new Business Partner Certificate.
1. After receiving the certificate file back, you need to bind the certificate to the private key that was generated in Step 1. This must be done on the same computer where the PowerShell script was run in Step 1. To bind the certificate, simply import it into the Local Computer certificate store. Note that if you import the certificate into the Current User certificate store, it will not bind to the private key. Windows will recognize the key container that was used to issue the certificate and will import the private key together with the certificate. After the certificate has been imported, you can then export the certificate—*with* its private key—to a .pfx file that you will install on your servers. If the import is unsuccessful and the certificate does not bind to the private key, repeat this process with a new key pair by running the PowerShell script again.

## Configuring Xbox Live Services access with Business Partner Certificates

To enable a server with a Business Partner Certificate to access Xbox Live Services, each title must grant access rights to the web service for which the Business Partner Certificate was generated. This configuration is performed in the “Service Configuration/Access” section of the title’s product instance on XDP.

1. To grant access for a service, first select the service scope (such as Stats, for example) and then select the **Edit** icon:


	1. Access to the selected service is granted by selecting “Add policy” and can be specified for either Read-Only or Full Access. This selection can also be further limited by service functionality.

	
	2. Select the web service from the principal list, and be sure to save the policy changes.
	3. Publish the product instance to have the policies created and made live on the service.

Note that this process has to be repeated for each web service that policies are created for. After a policy exists for a product instance, it will automatically copy forward when the product is copied or published to other sandboxes.

Removing an existing policy will prevent certificates created under that web service from being able to call Xbox Live Services in the context of that title. You can use this process if you want to block access because of a process change, server or certificate compromise, and so forth. You would need to remove the policy from all sandboxes (or copy and publish the change across any affected sandbox) to have the policy removed everywhere.

## Business Partner Certificates and sandboxes

If you are using a Business Partner Certificate without a sandbox specified in it (the recommended approach), the certificate will work across all of your development sandboxes as well as the Microsoft–managed sandboxes (such as CERT, CERT.DEBUG, RETAIL).

If you have sandbox-specific certificates, you will need a separate Business Partner Certificate for each of your development sandboxes and each Microsoft–managed sandbox. You must also ensure that your service uses the correct certificate for the sandbox that the title is running in when it calls Xbox Live Services. In most cases, using a certificate for a different sandbox than the one specified in the token from the title will result in an error from Xbox Live.

To avoid using the incorrect certificate for the current environment, the best approach is to dynamically detect which certificate is needed for that environment. All certificates are present on the server, and the client identifies which sandbox it is using. The safest way to do this is to use the Sandbox claim in the Xbox Security Token Service (XSTS) token and parse this information on the server. The server can then dynamically select the correct certificate based on the Sandbox string.

## Additional configuration to call Xbox Live on behalf of a user: delegation

To make service-to-service calls to Xbox Live on behalf of a user, your service must already be configured for single sign-on with Xbox Live. That is, it must be able to receive and process X tokens from clients (Xbox One or other clients in the future) and extract user claims from those tokens.

Assuming that your service is configured in this way, you must also ensure that the token definition for your relying party includes the “DelegationToken” claim. This claim contains the information needed by the XSTS to issue tokens on behalf of the user. You can verify or update your relying party definition from the Manage/Web Services page in XDP. Under the web service that contains the endpoints your title calls, select an endpoint configured for the relying party in question. Select the **Pencil** icon to edit the token definition, and add the DelegationToken claim if it is not already there.

**Note** Changes to relying parties and token definitions are global and apply to all titles or services that are configured to use this token. These changes also apply across all sandboxes. After making the change, be sure to publish your changes from the Manage/Web Services page.

### Business Partner Certificate expiration

Like any other certificate, Business Partner Certificates have an expiration time and should be updated before expiration. Expired Business Partner Certificates will not immediately fail, but may be blocked at some point.

Developers are encouraged to track the expiration date of any Business Partner Certificates in use and refresh them through XDP before the expiration date approaches.

# Service authentication flow: overview

The following diagram depicts the three steps that are involved in authenticating your service with Xbox Live Services.



1. First, your service performs an authentication request to the Xbox Authentication Service for Services (XASS[[1]](#footnote-2)). When making that call, the Business Partner Certificate obtained earlier needs to be used as a client SSL certificate. For a successful request, XASS will return an S (Service) token, which has a default lifetime of two weeks.
2. The S token can then be exchanged for an authorization token called an X token. This is done by making a request to XSTS,[[2]](#footnote-3) passing in the S token, the ID of the sandbox that you’re trying to access, and the name of the relying party you are requesting a token for as the properties of the request.
3. Optionally, you can add to the request the DelegationToken claim in case the call needs to be made on behalf of the user for whom you obtained that claim. In such a scenario, the supplied **SandboxId** must be accessible by the user of the DelegationToken claim.

Using the X token(s) thus obtained, your service can make authenticated calls to various Xbox Live services.

# Proof keys

Use of proof keys prevents a man in the middle from capturing tokens and authoring messages on behalf of the true token owner.

More specifically, when obtaining a security token, the caller first generates a public/private key pair. The caller then keeps the private key for itself and sends the public key with the security token request (explained in a later section of this document, [Obtaining a service token](#_Obtaining_a_service)). The public key is then embedded in the token. Each time the caller sends a request to a service, it passes the token to the service and signs the message with the private key—which only the caller has. The service receiving the message verifies the validity of the token and then validates the signature of the request by using the public key contained in the token. If the request signature validates, it proves that the caller is in possession of the private key. Assuming that the caller has kept its private key safe, this proves that the caller is the same party that provided the public key to the security token service.

The key itself is a JSON Web Key as defined in the following IETF specification: <http://tools.ietf.org/html/draft-ietf-jose-json-web-key-08>

The URL, path, query string, **Authorization** header, and timestamp are always signed. Thus, they are implicitly in the signature policy to be used to sign the request. Each service that exposes authenticated endpoints can require additional headers that must be included in the signature, and can also specify the maximum size (in bytes) of the request body required to sign. Each service defines the set of signing algorithms it supports.

The signature policy looks like this in JSON form:

{

"Version": 1,

 "SupportedAlgorithms": [ "ES256", "ES384" ],

 "ExtraHeaders": [ ],

 "MaxBodyBytes": 8192

}

Clients—your title service, in this case—need to learn about the required signing policy for the endpoints they need to call. They create a data stream to sign from the request (the exact process is described in Steps 1 through 7, below), hash the data stream by using Secure Hashing Algorithm 256 (SHA256), and then encrypt it by using the private key to produce a signature.

The signature is transmitted through an **HTTP** header and is made of three parts. The first part is the signature policy version and is expressed as a 4-byte integer value in network-byte order (big-endian). The second part is a 64-bit Windows file time[[3]](#footnote-4) in network-byte order. The third part is the resulting signature bytes. These three parts are combined as a single byte stream and encoded as a base-64 string that is sent with the request as an additional header—the **Signature** header.

Because the signature is transported as an **HTTP** header, any message that includes a hash of the body must be hashed in memory before any bytes of the body are written to the request stream. The signature is computed over the actual bytes sent over the wire. Obviously, the signature must be computed after any request-body transformations.

The exact process of creating the data stream to sign is shown below; all string data, such as the headers and URL, are encoded in ASCII. The body is an opaque blob and is signed as-is.

UINT policy version

0

TIMESTAMP time

0

STRING httpMethod

0

STRING pathAndQueryString

0

STRING authHeader

0

STRING otherHeaders

0

BYTES adjustedBody

0

The values of these elements are explained as follows:

1. The policy version as a 4-byte unsigned integer in network-byte order, followed by a null byte.
2. The timestamp (from the **Signature** header) in network-byte order, followed by a null byte.
3. The HTTP method, in all caps (GET or POST, for instance), followed by a null byte.
4. The absolute path and query string, followed by a null byte.
5. The value of the **Authorization** header (if present), followed by a null byte.
6. The other headers specified in the signature policy, each followed by a null byte if you’re adding any header. If there are no additional headers, no byte should be added.
7. Let N be the maximum number of bytes of the body to sign, as specified by the signature policy for the endpoint to be called. Let M be the size of the uncompressed body. If M > N then use the first N bytes of the uncompressed body. If M <= N then use the entire uncompressed body (without padding). A null byte is appended (the null byte must be appended even if there is no request body).

Each of the elements above is separated by a null byte (0x00) to mitigate any ambiguity from concatenation without a separator. The null byte is used because the HTTP standard forbids the use of non-printable characters in the headers. Thus, any attempt to insert or extend the data would need to include the null byte, and would thus be an invalid HTTP request.

The absolute path and query string are obtained from the URI of the request that is signed. If the URI of the request was “https://service.xbox.com/service1/foo?q0=v0&q1=v1#frag,” the absolute path and query string would be “/service1/foo?q0=v0&q1=v1#frag”. Note that the absolute path must start with a “/”.

Any headers in the request that are specified in the policy must be included in the signature in the same order as they appear in the signature policy. The **Authorization** header is always signed. If the policy specifies a header that is not present in the request, it does not have to be in the signature. For example, if the policy specifies headers [ H1, H2, H3 ] and the request includes only H1 and H2, then H3 can be left out. This is needed if the global policy specifies headers such as **Range**, which does not apply for POST requests. Note, however, that if a header is missing, the corresponding null bytes must still be included. This also applies if no headers are specified at all. So the data stream to sign would include two null bytes between the H1 and H3 headers if H2 were missing. Another way to think of this is to consider missing headers as an empty string.

# Obtaining a service token

Details regarding the request, response, and error handling associated with obtaining a service token are explained in this section.

## Request

XASS provides the following REST endpoint to handle service authentication:

<https://service.auth.xboxlive.com/service/authenticate>

That endpoint requires a valid Business Partner Certificate to be used as a client SSL certificate in the request.

Your service performs a POST, with a body obeying the following **XASSRequest** data contract:

 [DataContract]

 public class XASSRequest

 {

 [DataMember(EmitDefaultValue = false)]

 public string RelyingParty { get; set; }

 [DataMember(EmitDefaultValue = false)]

 public string TokenType { get; set; }

 [DataMember]

 public PropertyBag Properties { get; set; }

 }

 [DataContract]

 public class PropertyBag

 {

 [DataMember(EmitDefaultValue = false)]

 public string ServiceToken { get; set; }

 [DataMember(EmitDefaultValue = false)]

 public string DelegationToken { get; set; }

 [DataMember(EmitDefaultValue = false)]

 public Ecc256ProofKey ProofKey { get; set; }

 }

 [DataContract]

 public class Ecc256ProofKey

 {

 [DataMember(Name = "alg", Order = 0)]

 public string Algorithm { get; set; }

 [DataMember(Name = "kty", Order = 1)]

 public string KeyType { get; set; }

 [DataMember(Name = "use", Order = 2)]

 public string Use { get; set; }

 [DataMember(Name = "crv", Order = 3)]

 public string CurveType { get; set; }

 [DataMember(Name = "x", Order = 4)]

 public string X { get; set; }

 [DataMember(Name = "y", Order = 5)]

 public string Y { get; set; }

 }

The only required property in the **PropertyBag** for the **XASSRequest** is the **ProofKey**. The **ProofKey** needs to be a JSON Web Key, as indicated in an [earlier section](#_Proof_keys) of this document.

Additionally, the following headers must be included in the request:

|  |  |
| --- | --- |
| Header name | Header value |
| x-xbl-contract-version | 1 |
| content-type | application/json |
| signature | Message signature computed following the specification in the section of this document on [proof keys](#_Proof_keys). The signing policy for the XASS service is as follows:{ Version = 1, ExtraHeaders = [ ], MaxBodyBytes = long.MaxValue,SupportedAlgorithms = new[] { "ES256" }} |

Following is the code for a sample request:

POST 'https://service.auth.xboxlive.com/service/authenticate'

 x-xbl-contract-version: 1,

 Signature: AAAAAQHPR6izYEzPeW1W5ghsfJP+Vzop0bEleqi6+XNG1eMt2htQr22W84Nku4y4fLqnryN1dFZF/0RuLD3UyY5U3uaBr37p+27TuA==,

 Content-Type: application/json,

 Content-Length: 242

{

 "Properties":

 {

 "ProofKey":

 {

 "alg":"ES256",

 "kty":"EC",

 "use":"sig",

 "crv":"P-256",

 "x":"G5lQkFZPAGDEKmd4BUdpinSWa8ptp8JrCvpNZu0t-I0",

 "y":"mqHWdo9l3cq99t4xdI2gqhzLpf984oNF9jYA4D5mfnc"

 }

 },

 "RelyingParty":"http://auth.xboxlive.com",

 "TokenType":"JWT"

}

**Note**  Ensure that curve points x and y are an even length when converting them from decimal representation to hexadecimal representation. Failing to do this can result in an incorrect conversion length that will be rejected by the service.

## Response

The data contract for the response appears below:

[DataContract]

public class XASTokenResponse

{

 [DataMember]

 public DateTime IssueInstant;

 [DataMember]

 public DateTime NotAfter;

 [DataMember]

 public byte[] Token;

}

Sample Response Body:

{

 "IssueInstant":"2014-03-24T21:56:33.31115Z",

 "NotAfter":"2014-04-07T21:56:33.31115Z",

 "Token":"eyJlbmMiOiJBMTI4Q0JiY…<truncated>…WxnIjoiUGagXRLVVC-L4\",

 "DisplayClaims":null

}

## Error handling

If the proof key signature is invalid, the server returns a 403 error.

**Note** For the SSL channel to be successfully established, the full trust certificate chain for the Business Partner Certificate needs to be installed on the client. The certificate chain is published and the appropriate certificates can be downloaded by opening the Business Partner Certificate, selecting the **Certification Path** tab, and viewing/downloading the individual certificates in question.

The following two certificates must be installed in the intermediate CA store of the server calling Xbox Live Services:

* Business\_Partner\_Certificate\_Issuing\_CA.cer
* Xbox\_Services\_RSA\_PCA\_2013.cer

The following certificate must be installed in the trusted root CA store of the server calling Xbox Live Services:

* Microsoft\_IEB\_Services\_RSA\_Root\_2013.cer

# Obtaining an X token

Details regarding the request, response, and error handling associated with obtaining an X token are explained in this section.

## Request

The XSTS service provides the following REST endpoint to handle authorization:

<https://xsts.auth.xboxlive.com/xsts/authorize>

Your service performs a POST, with a body obeying the following **XSTSRequest** data contract:

 [DataContract]

 public class XSTSRequest

 {

 [DataMember(EmitDefaultValue = false)]

 public string RelyingParty { get; set; }

 [DataMember(EmitDefaultValue = false)]

 public string TokenType { get; set; }

 [DataMember]

 public PropertyBag Properties { get; set; }

 public byte[] ProofKey { get; set; }

 }

Following is the code for a sample request:

POST https://xsts.auth.xboxlive.com/xsts/authorize HTTP/1.1

x-xbl-contract-version: 1

Signature: AAAAAQHPljBFa8IDokJK3DPInYd8yzJiQOw5dvhAwN9JEPjkqaC7PirhKpUuhhG1Bt3S9EGlYNlzDNQi0raKe0Swes/vpHQk6UT90w==

Content-Type: application/json

Host: xsts.auth.xboxlive.com

Content-Length: 6473

{

 "RelyingParty": "http://xboxlive.com",

 "TokenType": "JWT",

 "Properties": {

 "ServiceToken": "eyJlbmMiOiJBM**<truncated>**AY",

 "SandboxId": "XDKS.1"

 }

}

### The RelyingParty property

Tokens are issued (and encrypted) for specific relying parties. A service configured for a given relying party will only be able to consume tokens issued for that relying party.

Therefore, depending on the Xbox Live service that you need to call, you might need to retrieve different tokens, each time specifying the relevant relying party that you need a token for.

The following table indicates which relying party you need to get a token for in order to successfully access each Xbox Live service.

|  |  |
| --- | --- |
| Xbox Live Service HostName | Relying Party Name |
| https://musicdelivery-ssl.xboxlive.com | http://music.xboxlive.com |
| https://cloudcollection-ssl.xboxlive.com | http://music.xboxlive.com |
| https://music.xboxlive.com  | http://music.xboxlive.com |
| https://inventory.xboxlive.com | http://licensing.xboxlive.com |
| https://licensing.xboxlive.com | http://licensing.xboxlive.com |
| https://accountstroubleshooter.xboxlive.com | http://accounts.xboxlive.com |
| https://\*.xboxlive.com (if not listed above) | http://xboxlive.com |

It is also possible to retrieve a token for a custom relying party that is specified for a title endpoint (for example, http://mytitle.com/). Note that custom relying party names, unlike Xbox Live relying party names, have a trailing ‘/’.

### The PropertyBag property

The **PropertyBag** data contract for XSTS is as follows:

 [DataContract]

 public class PropertyBag

 {

 [DataMember(EmitDefaultValue = false)]

 public string ServiceToken { get; set; }

 [DataMember(EmitDefaultValue = false)]

 public string[] UserTokens { get; set; }

 [DataMember(EmitDefaultValue = false)]

 public string SandboxId { get; set; }

 [DataMember(EmitDefaultValue = false)]

 public string DelegationToken { get; set; }

}

The **ServiceToken** property should contain the *Token* value of the response from XASS.

The **SandboxId** property should contain the name of the sandbox that you’re trying to access, for example, “ABCD.1”. The main sandbox for all retail users and content is named “RETAIL” (note that the name is uppercase; it is case-sensitive).

**Note** For the case where the Business Partner Certificate you’re using has been issued for a specific sandbox, you must use the same value here.

The **DelegationToken** property is optional in this scenario, and should be used only if you’re calling Xbox Live on behalf of the user. Here you should include the value of the DelegationToken claim extracted from an X token that you have previously received from an Xbox One console or other client. The DelegationToken claim can be added to your XSTS tokens in the Xbox Developer Portal (XDP) as detailed in the section [Additional configuration to call Xbox Live on behalf of a user: delegation](#_Additional_configuration_to), earlier in this white paper.

The **UserTokens** property should be used only if you’re making calls to Xbox Live from a website where the web servers get authenticated with a Business Partner Certificate, and the user gets authenticated to your website by using the Microsoft account OAuth flow for user authentication. In that case, the **UserTokens** property should be an array of one element that contains the token retrieved by exchanging the user’s access token retrieved from the Microsoft account for a U (User) token at the XASU (Xbox Authentication Service for Users) service.

### Headers

|  |  |
| --- | --- |
| Header name | Header value |
| x-xbl-contract-version | 1 |
| content-type | application/json |
| signature | Message signature computed following the specification in the section of this document on [proof keys](#_Proof_Keyskeys).The same key must be used to sign this message as was used to sign messages to other authentication services (XASS or XASU).The signing policy for the XSTS service is as follows:{ Version = 1, ExtraHeaders = [ ], MaxBodyBytes = long.MaxValue,SupportedAlgorithms = new[] { "ES256" }} |

## Response

The data contract for the XSTS response is as follows:

 [DataContract]

 public class XSTSTokenResponse

 {

 [DataMember(Name = "IssueInstant", Order = 0)]

 public string IssueInstant { get; set; }

 [DataMember(Name = "NotAfter", Order = 1)]

 public string NotAfter { get; set; }

 [DataMember(Name = "Token", Order = 2)]

 public string Token { get; set; }

 [DataMember(Name = "DisplayClaims", Order = 3)]

 public XSTSDisplayClaims DisplayClaims { get; set; }

 public byte[] SigningProofKey { get; set; }

 }

 [DataContract]

 public class XSTSDisplayClaims

 {

 [DataMember(Name="xui")]

 public XuiClaims[] XuiClaims { get; set; }

 }

 [DataContract]

 public class XuiClaims

 {

 [DataMember(Name = "agg",EmitDefaultValue=false)]

 public string AgeGroup { get; set; }

 [DataMember(Name = "gtg", EmitDefaultValue = false)]

 public string Gamertag { get; set; }

 [DataMember(Name = "prv", EmitDefaultValue = false)]

 public string Privileges { get; set; }

 [DataMember(Name = "xid", EmitDefaultValue = false)]

 public string Xuid { get; set; }

 [DataMember(Name = "uhs", EmitDefaultValue = false)]

 public string UserHash { get; set; }

 }

If no **DelegationToken** or **UserTokens** properties were specified in the request, the response will not contain any display claims.

Otherwise, the **DisplayClaims** property of the response will contain a set of information about the user, as defined in the preceding data contract:

* **Agegroup**: this can be Child, Teen, or Adult.
* **Gamertag**: the gamertag of the user.
* **Privileges**: the set of privileges that the user has. (For more information about privileges, refer to the [Relying Party SDK documentation](https://developer.xboxlive.com/en-us/platform/development/downloads/Pages/home.aspx).)
* **Xuid**: the xuid of the user.

**Note** You must *not* store the user xuid in your databases unless you have been given express consent by Microsoft (through your DAM) to do so.

* **UserHash**: unique identifier for the user to be used when constructing **Authorization** headers for requests to Xbox Live Services. See [Calling Xbox Live Services](#_Calling_Xbox_Live) for more information.

Note that not all relying parties expose all **DisplayClaims**. To receive all claim members, use the <http://xboxlive.com> relying party name.

Following is the code for a sample response for the “http://xboxlive.com” relying party:

HTTP/1.1 200 OK

Cache-Control: no-cache, no-store

Content-Length: 3196

Content-Type: application/json

X-Content-Type-Options: nosniff

X-XblCorrelationId: bafef442-9a66-4351-ae48-ef1812295b8e

Date: Wed, 02 Jul 2014 20:00:29 GMT

{
   "IssueInstant":"2014-07-02T20:00:29.3191631Z",
   "NotAfter":"2014-07-03T04:00:29.3191631Z",
   "Token":"eyJlbmMiO**<Truncated>** FtM",
   "DisplayClaims":{
      "xui":[
         {
            "agg":"Adult",
            "gtg":"Cool Gamertag here",
            "prv":"190 191 193 194 196 198 199 200 201 203 204 205 206 207 208 209 214 217 220 224 227 228 235 238 245 247 249 250 252 254 255",
            "xid":"2814630418365389",
            "uhs":"1283950176146904870"
         }
      ]
   }
}

## Error handling

If the XSTS token request is rejected, the response will, in some cases, contain data indicating why it was rejected.

 [DataContract]

 public class AuthorizeResponseNotAuthorized

 {

 [DataMember(Name = "Identity")]

 public string Identity { get; set; }

 [DataMember(Name = "XErr")]

 public uint XErr { get; set; }

 [DataMember(Name = "Message")]

 public string Message { get; set; }

 }

The **Identity** and **Message** properties can be ignored.

The **XErr** property can have the following values:

|  |  |
| --- | --- |
| Value | Description |
| 0x8015DC03 | There is an issue with the user account. The user should be advised to resolve any issue either on the console or by signing in to https://xbox.com. |
| 0x8015DC05 |
| 0x8015DC09 |
| 0x8015DC0A |
| 0x8015DC0B |
| 0x8015DC0C |
| 0x8015DC0D |
| 0x8015DC0E |
| 0x8015DC0F |
| 0x8015DC10 |
| 0x8015DC13 |
| 0x8015DC12 | Access to the sandbox specified in the request was denied. You should verify that the correct sandbox was specified in the request, and/or that the appropriate access policies were created through your DAM. |
| 0x8015DC1F | An expired service token was passed in the request. |
| 0x8015DC22 | An expired User token was passed in the request. |
| 0x8015DC26 | An invalid User token was passed in the request. |
| 0x8015DC27 | An invalid service token was passed in the request. |
| 0x8015DC31 | Xbox Live authentication infrastructure is currently experiencing an outage. |
| 0x8015DC32 |

# Calling Xbox Live Services

The objective of obtaining X tokens from the XSTS service is to make authenticated calls to various Xbox Live Services.

For that to be successful, an **Authorization** header must be specified when making calls to these services. The **Authorization** header takes the following structure:

authorization: XBL3.0 x=<userHash>;<XToken>

The *userHash* part of that header should be set as follows:

* For the case where a **DelegationToken** or **UserTokens** property was set in the XSTS request, the **UserHash** property from the *DisplayClaims* element of the XSTS response should be used.

Example:
XBL3.0 x=1077552597660441275;eyJlbmMiOiJBMTI4Q0JDK…(truncated)

* For the case where only a **ServiceToken** was specified in the XSTS request, the **UserHash** property should be set to the character “-“.

Example:
XBL3.0 x=-;eyJlbmMiOiJBMTI4Q0JDK…(truncated)

The *XToken* part of that header should be set to the value of the *Token* element in the XSTS response. Note that X tokens have an expiration date and time, as specified in the *NotAfter* element of the XSTS response. If an expired token is used to call Xbox Live Services, the request will be denied with an HTTP status of 401 and an indication in the www-authenticate header of the response that the token was expired.

Xbox Live Services also require a message signature computed following the specification [described earlier.](#_Proof_keys)

When calling Xbox Live Services from your service, be sure to reuse existing connections when possible instead of establishing new connections for each request. This will increase throughput and avoid request throttling.

## Calling Xbox Live Multiplayer services

Calls to the Xbox Live Multiplayer service vary from other Xbox Live Services calls. To correctly call the Multiplayer Session Directory (MPSD) and SmartMatch service, a Business Partner Certificate is required, but calls with a Delegation token are not supported.

Additionally, a full-access **Multiplayer.Manage** policy must be in place for the web service that the Business Partner Certificate was created under. Thereafter, the server will need to:

1. Call XASS with the Business Partner Certificate to retrieve an S token as outlined previously.
2. Call XSTS with this S token and a **SandboxId** to receive an X token. A Delegation token or User token is no longer required in this step. When specifying a **SandboxId** through this flow, the same sandbox must be specified during the creation of the Business Partner Certificate. Business Partner Certificates without a specific sandbox are not supported and will cause authentication errors that indicate a missing sandbox.
3. Call the MPSD service and/or SmartMatch service with the X token and headers (as specified in the next section).

### Xbox Multiplayer Session Directory headers

**Title header**

To act as a particular title, the **X-Xbl-OnBehalfOf-Title** header is required in the following format:

X-Xbl-OnBehalfOf-Title:[titleid in decimal]

*Example:*

Request.Headers["X-Xbl-OnBehalfOf-Title"] = "484921321";

This header must be specified to make calls for a particular title.

**User header**

To act as a particular user or set of users, the **X-Xbl-OnBehalfOf-Users** header is required in the following format:

X-Xbl-OnBehalfOf-Users:[xuid][;privilege][,xuid[;privilege]]...

*Example:*

Request.Headers["X-Xbl-OnBehalfOf-Users"] = "741837829132;priv=multiplayer";

The only privilege currently supported is “priv=multiplayer,” which indicates that the user has the multiplayer privilege. In addition, the server can act as only one user.

When the user header is specified, the standard “me” member in a session document body cannot be used. Instead, the following three options are available:

{

    "members": {

        // (e.g: me\_59135345328) Requires a user principal with a xuid claim. Can be

 'null' to remove myself from the session.

        "me\_{xuid}": {

            "constants": { /\* Property Bag \*/ },

            "properties": { /\* Property Bag \*/ },

        },

        // Applies the requested change to each acting user's member. Can be 'null' to

 remove acting users from the session.

        "me\_all": {

            "constants": { /\* Property Bag \*/ },

            "properties": { /\* Property Bag \*/ },

        },

        // Applies the requested change to each acting user's member if they are

 already in the session. Can be 'null' to remove acting users from the

 session.

        "me\_allInSession": {

            "constants": { /\* Property Bag \*/ },
            "properties": { /\* Property Bag \*/ },
        }

    }

}

Both the user and title header can be specified at the same time. Services should use these headers to optimize calling patterns by merging multiple MPSD calls for different users into a single MPSD call.

When using the **X-Xbl-OnBehalfOf-Users** header, it is as though the user identified in the header is making the call directly from his or her console. Because of this, the calling service is required to maintain the user’s security.

* Only real xuids can be used.
* The privileges declared for the user must be correct.
* The user must have consented to any action taken on his or her behalf.

The last requirement means that the service is permitted to take only actions that the console title could have taken itself.

For example, a service can set a user as active in a title only if the user is, in fact, running and interacting with that title on the console. The service must then set the user to inactive when he or she is no longer interacting with the title on the console. Similarly, a service can send an invite on behalf of a user only if the user has taken an explicit action to send the invite.

**Deny-Scope header**

The Multiplayer.Manage access policy overrides user access permissions for service-to-service calls to the MPSD service. Any session access is therefore not restricted based on user permissions. To re-enable user permission checks, you can use the **X‑Xbl-Deny-Scope** header:

X-Xbl-Deny-Scope: Multiplayer.Manage

*Example:*

Request.Headers["X-Xbl-Deny-Scope"] = "Multiplayer.Manage";

This header will ensure that the Multiplayer.Manage access policy is not used as an override when checking a user’s access (from the user header) to a session. It can be used to ensure that a user has the correct access to a session and that the server access is not overriding any other blocks because of visibility or join restrictions.

When setting this header, the Multiplayer.Runtime access must also be granted for the web service as a fall-back. This allows access even when user permissions are denied. Otherwise, there is no access at all in an error scenario and a 403 status will be returned. Multiplayer.Runtime access requires an acting user, so the **X-Xbl-Deny-Scope** header will only work together with the **X‑Xbl-OnBehalfOf-Users** or a user claim obtained from a DelegationToken claim.

### Xbox SmartMatch headers

The Xbox SmartMatch service verifies user information against the session. When the server calls on behalf of a user, however, it simply acts as the session’s first user and is automatically granted a privilege to match tickets as long as the Multiplayer.Manage policy is set on the Business Partner Certificate.

However, there is a special case if a Peer-to-Peer QoS rule is configured for a matchmaking hopper. Normally, for QoS prediction between peers, SmartMatch will use the client’s IP address. Because the server’s IP address is not useful, the server must provide the client’s IP address in order for SmartMatch to predict the QoS appropriately.

This is specified with the **X-Xbl-Server-Assigned-IP** header as follows:

X-Xbl-Server-Assigned-IP:[IP address]

*Example:*

Request.Headers["X-Xbl-Server-Assigned-IP"] = "10.124.172.137";

If the header is not specified, the SmartMatch service will use a default value, which greatly reduces the usefulness of the QoS rule.

# References

* [Xbox Services and Relying Party SDK](https://developer.xboxlive.com/en-us/platform/development/downloads/Pages/home.aspx)
* [Understanding Security Tokens for Xbox One](http://aka.ms/9837856) (white paper)
* [XDK Documentation](https://developer.xboxlive.com/en-us/platform/development/documentation/software/Pages/home.aspx): API Reference/Live and Online API Reference/Live Services RESTful Reference
* [Security Token Service Issuer Certificates](https://developer.xboxlive.com/_layouts/xna/download.ashx?file=xsts.auth.xboxlive.com.cer.zip&folder=platform\RelyingParty)
* [Your Xbox One Title, XSTS Tokens, and Web Services](http://aka.ms/9830307) (white paper)
1. <https://service.auth.xboxlive.com/service/authenticate> [↑](#footnote-ref-2)
2. <https://xsts.auth.xboxlive.com/xsts/authorize> [↑](#footnote-ref-3)
3. <http://msdn.microsoft.com/en-us/library/system.datetime.tofiletimeutc.aspx> [↑](#footnote-ref-4)