

User Manual

# Ayla Single Sign On (SSO)

---

Approach, API, Code Samples, and Troubleshooting



Version: 4.0

Date Released: May 8, 2018

Document Number: AY006USS6-4



---

## Copyright Statement

© 2018 Ayla Networks, Inc. All rights reserved. Do not make printed or electronic copies of this document, or parts of it, without written authority from Ayla Networks.

The information contained in this document is for the sole use of Ayla Networks personnel, authorized users of the equipment, and licensees of Ayla Networks and for no other purpose. The information contained herein is subject to change without notice.

---

## Trademarks Statement

Ayla™ and the Ayla Networks logo are registered trademarks and service marks of Ayla Networks. Other product, brand, or service names are trademarks or service marks of their respective holders. Do not make copies, show, or use trademarks or service marks without written authority from Ayla Networks.

---

## Referenced Documents

Ayla Networks does not supply all documents that are referenced in this document with the equipment. Ayla Networks reserves the right to decide which documents are supplied with products and services.

---

## Contact Information

### Ayla Networks TECHNICAL SUPPORT and SALES

Contact Technical Support: <https://support.aylanetworks.com>  
or via email at support@aylanetworks.com

Contact Sales: <https://www.aylanetworks.com/company/contact-us>

### Ayla Networks REGIONAL OFFICES

#### GREATER CHINA

Shenzhen  
Room 310-311  
City University of Hong Kong  
Research Institute Building  
No. 8 Yuexing 1<sup>st</sup> Road  
High-Tech Industrial Park  
Nanshan District  
Shenzhen, China  
Phone: 0755-86581520

#### HEADQUARTERS

Silicon Valley  
4250 Burton Drive, Suite 100  
Santa Clara, CA 95054  
United States  
Phone: +1 408 830 9844  
Fax: +1 408 716 2621

#### EUROPE

Munich  
Ludwigstr. 8  
D-80539 München  
Germany

#### TAIWAN

Taipei  
5F No. 250 Sec. 1  
Neihu Road, Neihu District  
Taipei 11493, Taiwan

#### JAPAN

Room #701  
No. 2 Ueno Building 3-7-18  
Shin-Yokohama, Kohoku Ward  
Yokohama City, 222-0033 Japan  
Phone: 045-594-8406

For a complete contact list of our offices in the US, China, Europe, Taiwan, and Japan, click:

<https://www.aylanetworks.com/company/contact-us>

## Table of Contents

1	Introduction.....	1
1.1	Audience .....	1
1.2	Revision History .....	1
2	About SSO Development.....	3
2.1	External Identity Providers .....	3
2.2	Goals.....	3
2.3	Approach.....	3
2.3.1	Convention.....	3
2.3.2	Interactions.....	4
2.4	Sequence Diagrams.....	5
3	Identity Provider Requirements .....	8
3.1	Provider Authentication.....	8
3.1.1	Client Logic.....	8
3.1.2	Provider Logic.....	11
3.2	Ayla Authentication .....	12
3.3	API Specification.....	12
3.3.1	Token Validation API.....	12
3.3.2	User Profile Format .....	14
3.3.3	Get User Profile API .....	15
3.3.4	Update and Delete Accounts.....	16
3.3.5	Login with Identity Provider Access Token.....	18
3.3.6	Add Your IDP .....	20
4	Curl Examples & Explanation .....	23
4.1	User Creation at Initial Auth.....	23
4.2	Token validate .....	23
4.3	Get User Profile.....	24
4.4	Update User .....	25
4.5	Delete User .....	25
4.6	Timeout.....	26

5	Setup .....	27
5.1	Client Signing Sample Code (Ruby).....	27
5.2	Provider Verification Sample Code (Ruby) .....	30
6	Troubleshooting .....	35
6.1	Example 1 .....	36
	INPUTS:.....	36
	COMPUTED VALUES: .....	36
	STRING TO SIGN:.....	37
	SIGNING KEY:.....	37
	AUTH HEADER: .....	37

# 1 Introduction

This document provides the Ayla platform interfaces to integrate with external identity providers for user account management

## 1.1 Audience

---

This document is written for engineers who are familiar with web services.

## 1.2 Revision History

---

Version	Date	Author	Comments
2	29 Feb 2016	R. Breger, L.Boling	Draft update – Example for DELETE and new x-ayla-origin-host, corrected status codes, replaced incorrect 2 curl examples; fixed key_timestamp
3	05 April 2016	D.Fulton	Modified Signed Headers, Identity Provider Requirements and User Profile Format
4	Mar 2017		Migrated to new document template and minor edits



## 2 About SSO Development

This section provides details on how to set up Single Sign On (SSO) in cooperation with Ayla Networks.

### 2.1 External Identity Providers

---

OEMs often have existing systems to manage users for support and tracking purposes. With a partner service such as Ayla, OEMs prefer to handle authentication against their systems. This helps users avoid the need to sign up against multiple services. Federated SSO (SAML, OAuth) is the standard way for web applications to achieve Single Sign-On.

However, this does not work for all Ayla customers for several reasons:

- The Ayla service and the OEM service have a partner relationship, and to the consumer should appear as one service. Typical SAML and OAuth flows present the two services as two distinct entities to the end user.
- Both SAML and OAuth rely on web redirects, which is a less desirable user experience from native apps.
- A standard OAuth flow requires explicit authorization from the user, which can be confusing for an end user, since the app is already owned by the OEM

### 2.2 Goals

---

- Allow consumers to authenticate with an OEM's existing user management system
- Offer a seamless account sign up/ sign in experience from consumer apps
- Avoid multiple sign-up, sign-in and authorization prompts for end users
- Avoid web redirects for a smooth user experience

### 2.3 Approach

---

The Ayla User Service is responsible for user management within the Ayla platform. The OEM service is treated as an identity provider with delegated user authentication. On successful authentication from the identity provider, the Ayla User Service allows the user to continue using its platform services.

#### 2.3.1 Convention

Identity Provider - service responsible for user management and primary repository of user account information.

End User App - application consumers and other end users use to interact with Ayla-enabled devices

Ayla Service - Ayla user service responsible for user account management within the Ayla platform. This service acts as a client to the identity provider.

### 2.3.2 Interactions

---

#### User Creation

The user account is originally created with the Identity Provider, and the provider is responsible for confirming the user's email and account setup.

---

#### Initial User Login

The first user login with the Ayla service accepts an access token issued by the provider, and verifies the validity of the token with the provider. On successful validation, the provider returns a user profile that creates the user entity in the Ayla service.

Subsequently, the Ayla service returns a short-lived auth token (currently 1 day validity) and a long-lived refresh token (currently six-month validity). The auth token can be used for further interactions with the service.

---

#### Subsequent logins

As part of the initial login, the Ayla service returns a refresh token to obtain a new auth token without requiring the user to log back in.

If the Ayla refresh token expires, the app cannot renew its Ayla auth token. It can take two different paths:

- Use the refresh token issued by the identity provider to renew the provider access token (provided the token is valid). It can then present the new provider access token to Ayla.
- If provider refresh token is not valid, the user falls back to the initial login flow (app prompts user to log in again).

---

#### User Profile Updates

In the event of a user profile update with the identity provider, the provider invokes an Ayla URL pre-registered as part of the initial setup. This triggers a call to the provider to fetch the updated user profile. See [Setup](#).

---

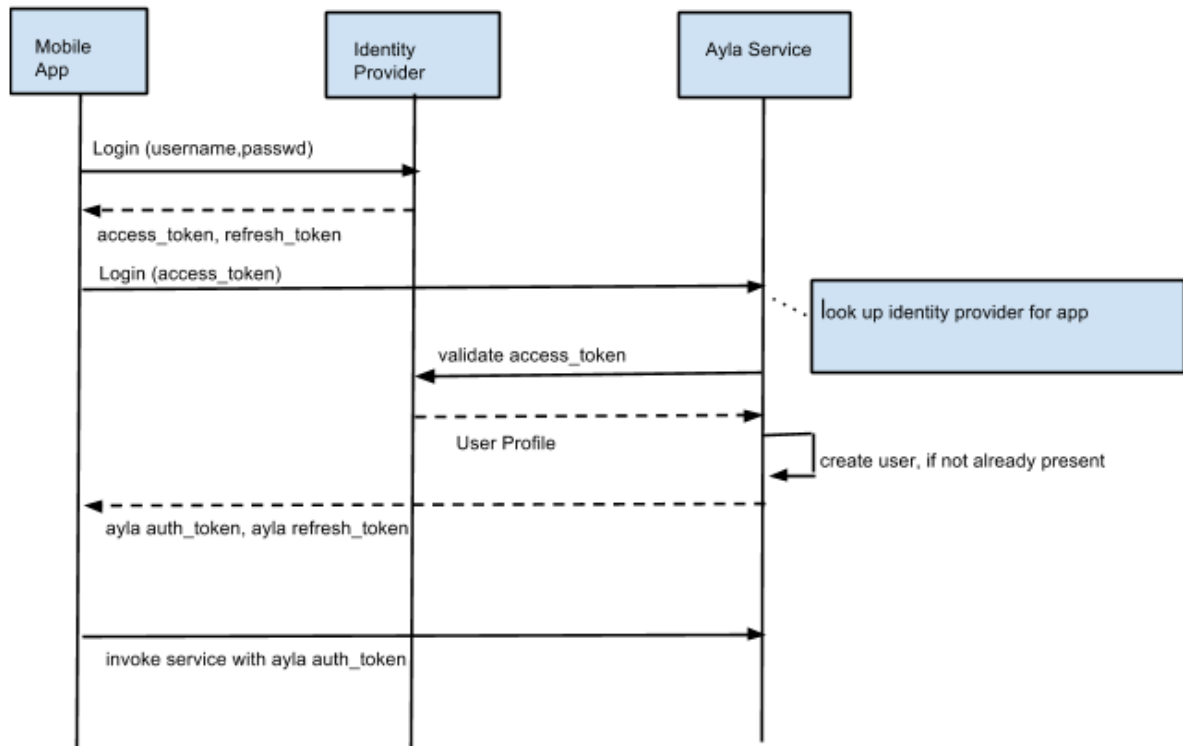
#### User Account Deletion

In the event of a user canceling their account with the identity provider, the provider invokes a pre-registered Ayla URL (part of the initial setup). This triggers a call to the provider to fetch the updated user profile. If no valid response, the user entity is deleted on the Ayla service, and further interactions are not be allowed for the user. See [Setup](#).

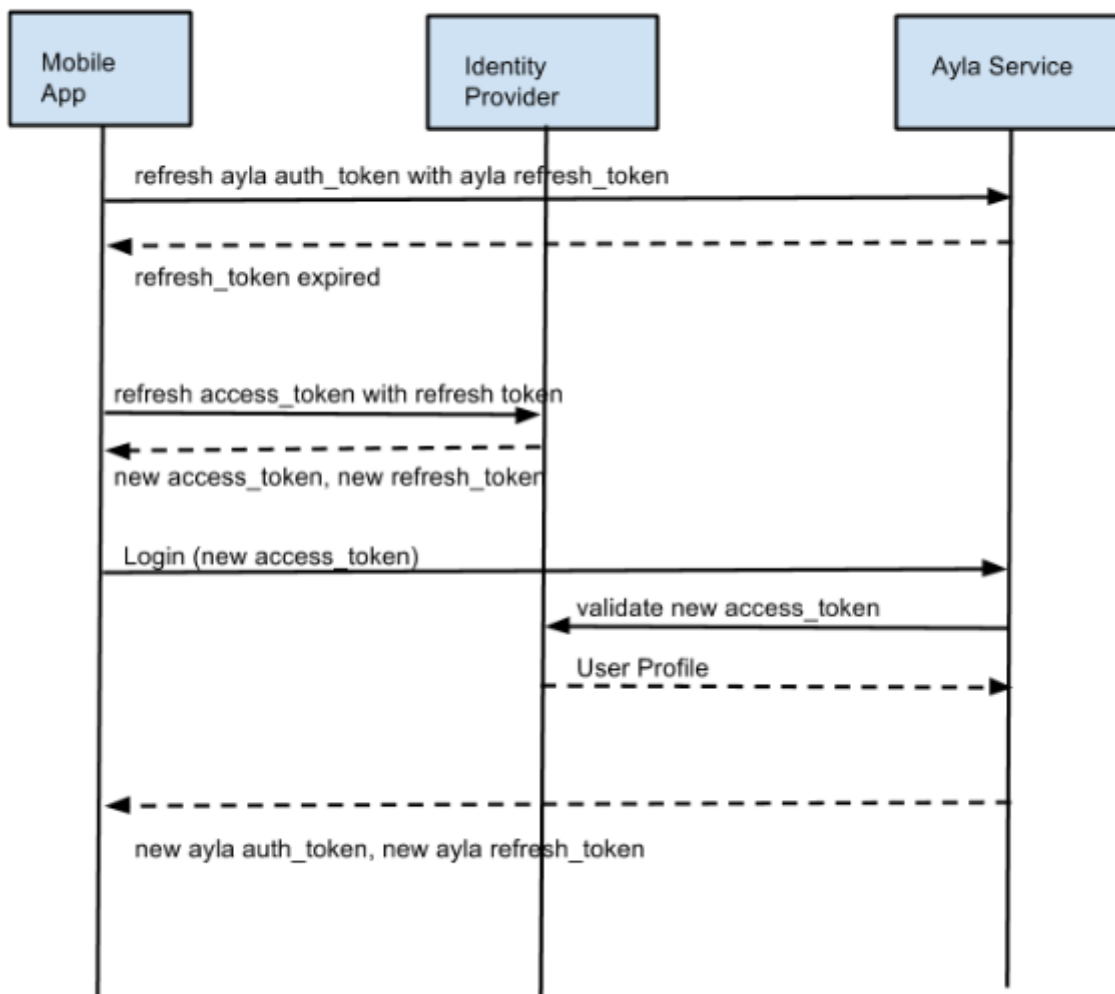


## 2.4 Sequence Diagrams

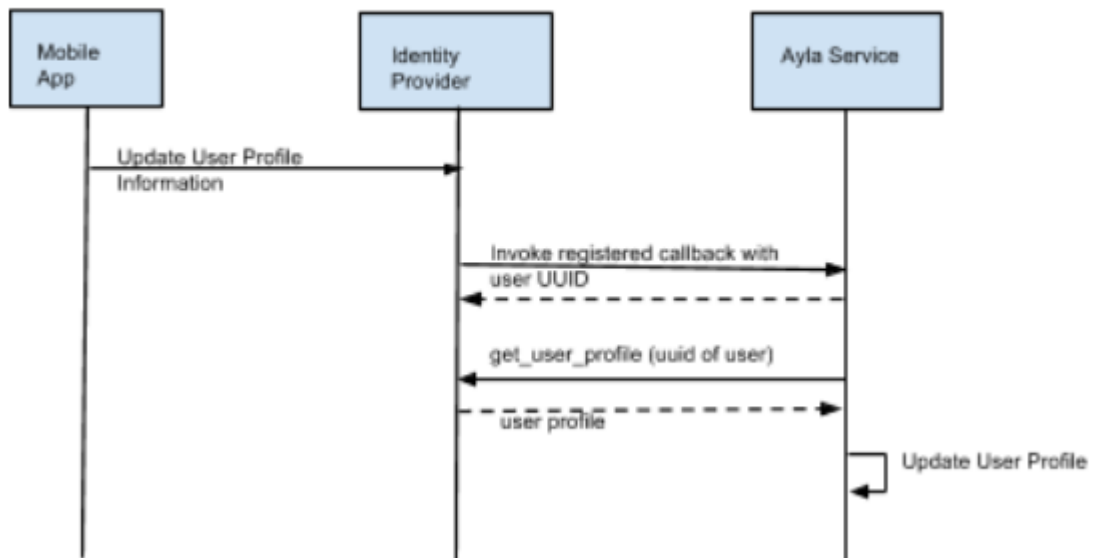
Sequence: User Sign-In with Ayla



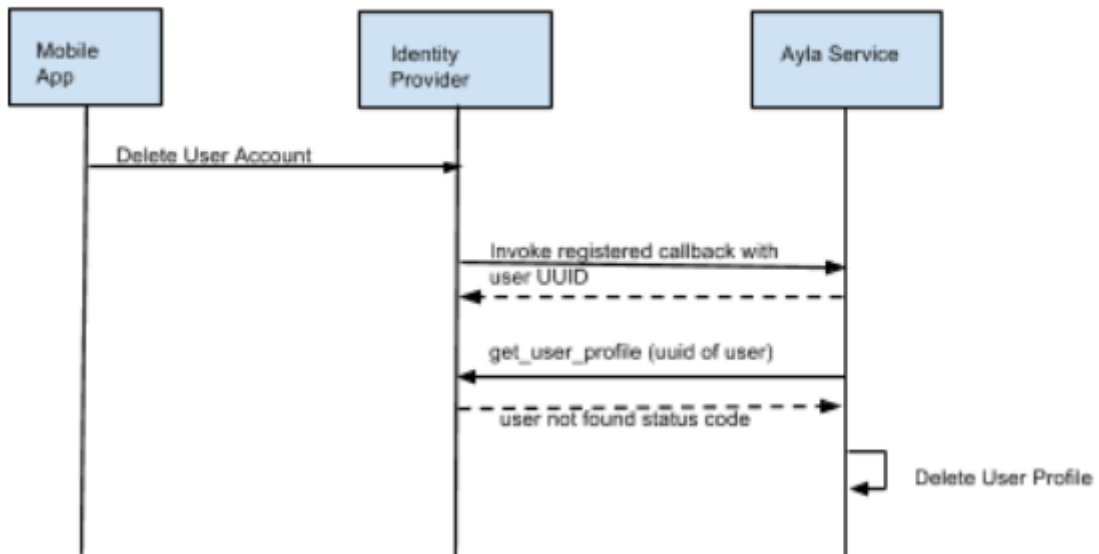
## Sequence: Refreshing access token after Ayla refresh token expires



### Sequence: User Profile Changes



### Sequence: User Account Deletion



## 3 Identity Provider Requirements

This section specifies requirements for an external service to become an Identity Provider with Ayla.

1. The provider specifies a provider app id and provider app secret for its clients to authenticate.
  - The provider app id identifies the specific client that performs authentication
  - The provider app secret, used by the client, generates a signature that authenticates it with the provider. The secret key should be a cryptographically secure random number. See [Provider Authentication](#)
2. The provider specifies a service endpoint (URL) to:
  - Implement a GET method
  - Accept an access token in a URL parameter-called token
  - Return the user profile if the token is valid. See [User Profile Format](#)
3. The provider specifies a service endpoint (URL) to:
  - Implement a GET method
  - Accept a user identifier in a URL parameter-called *uuid*
  - Return the user profile if token is valid, or return an error code if uuid is not valid. See [User Profile Format](#)

See error code format specified in [API Specification](#)
4. The provider allows clients to register callbacks for user profile update and deletion events. The provider invokes these callbacks in accordance with the specification in [Update and Delete Accounts](#).
5. All calls to the identity provider must return within 15 seconds.

### 3.1 Provider Authentication

This section specifies a signature-based authentication method to verifies client identity to the provider.

The client performs these steps to generate a signature with each API request to the provider. The provider verifies the signature with the same methods to authenticate the client.

#### 3.1.1 Client Logic

Follow this process:

---

##### Step 1: Create a canonical request

1. Define the HTTP verb.

For the APIs specified in this document, the provider only implements a GET method. The HTTP verb is always "GET".

2. Determine the UTC time in ISO8601 format. This is sent in the x-sso-date header.

Example: 20150817T063855Z

3. Extract the value of x-ayla-origin-host header. This creates the canonical header.

Example: provider.com

4. Define the canonical URI.

If the URL is "[https://provider.com/apiv1/is\\_valid\\_token](https://provider.com/apiv1/is_valid_token)", the canonical URI is `/apiv1/is_valid_token`

5. Create the canonical query string.

- For each request parameter, URL encode the value. Arrange parameters alphabetical by name. Append the character '=' after every parameter name, followed by the URL-encoded parameter value. Append the character '&' after every parameter value, except the last one.

- Example: Given this URL:

[https://provider.com/apiv1/is\\_valid\\_token?token=9b54CXk/OCL1U8m+qXc&context=some%20context](https://provider.com/apiv1/is_valid_token?token=9b54CXk/OCL1U8m+qXc&context=some%20context)

- For the above example, the canonical query string is:

`context=some%20context&token=9b54CXk/OCL1U8m+qXc`

Notice that the "context" parameter comes before the "token" parameter, in alphabetical order.

6. Create the list of signed headers.

This list of headers is included in the signature, sorted alphabetical by name and separated by ";".

"x-ayla-origin-host" and "x-sso-date" are mandatory.

The signed header is `"x-ayla-origin-host;x-sso-date"`

7. Create the canonical header.

For each header in the list of signed headers, append a header name, followed by a colon and space, followed by the header value. Insert "\n" after each header value.

The x-sso-date header value should be a timestamp with the **ISO 8601** format.

Example: Given these headers:

`x-ayla-origin-host: provider.com`

`x-sso-date: 20150817T063855Z`

Canonical header for the above example is:

`x-ayla-origin-host: provider.com\n`

`x-sso-date: 20150817T063855Z\n`

---

## 8. Create the canonical request.

Combine all the elements described above

```
http_verb + '\n' + canonical_uri + '\n' + canonical_query_string + '\n' + canonical_headers + '\n' + signed_headers
```

Example:

```
"GET" + '\n' + "/is_valid_token" + '\n' + "context=some%20context&token=9b54CXk/OCL1U8m+qXc" + '\n' + "x-ayla-origin-host: provider.com\n\nx-sso-date: 20150817T063855Z\n" + '\n' + "x-ayla-origin-host;x-sso-date"
```

---

## Step 2: Create a string to sign

### 1. Define the scope string.

The scope string can be specified by the provider as part of the setup. By default "user/sso/v1"

### 2. Determine the signing algorithm. It is "HMAC-SHA256".

### 3. To create the complete string to sign, combine the following algorithm + '\n' + time\_in\_iso\_8601\_format + '\n' + scope\_string + '\n' + canonical\_request.

Example:

```
"HMAC-SHA256" + '\n' + "20150817T063855Z" + '\n' + "user/sso/v1" + '\n' + <canonical_request_from_step_1>
```

---

## Step 3: Create a signing key

### 1. Define a salt.

- The salt can be configured by the provider as part of the setup.
- Default value for the salt is "AYLA-SSO"

### 2. Add the salt to the provider app secret key and encode it in UTF-8.

### 3. Create an HMAC signature of the timestamp, signed with the above string

---

## Step 4: Create a signature

### 1. UTF-8 encode the string to sign, computed in Step 2.

### 2. Create an HMAC signature for the string to sign with the signing key, computed in step 3

### 3. Hexencode the binary HMAC signature, with lowercase characters.

---

## Step 5: Create an authorization header

### 1. To create an authorization header, combine:

```
authorization_header="Authorization: " + algorithm + " " + "Credential=" + provider_app_id + "/" +  
scope + ", " + "SignedHeaders=" + signed_headers + ", " + "Signature=" + signature
```

## 2. Example:

```
Authorization: HMAC-SHA256 Credential=provider-id/user/sso/v1, SignedHeaders=x-ayla-origin-  
host;x-sso-date,  
Signature=0bb5f8614cec9270b7f2bcf0c2c6bc4739258dcb486429d58d7c8e69c93e0964
```

---

## Step 6: Make the request to the provider

1. Initiate the GET request with the Authorization header.
2. Include the x-sso-date header and x-ayla-origin-host header.

### 3.1.2 Provider Logic

The Identity Provider uses the following steps to authenticate the client request.

---

## Step 1: Determine provider parameters

*Method:* GET

*Scope:* String identifying the scope of the authentication. By default "user/sso/v1"

*Salt:* String added to the provider app secret to derive the signing key. By default "AYLA-SSO"

*URI:* The URI part of the request

*Server Time:* Timestamp on the provider server in UTC

---

## Step 2: Extract request parameters

1. Extract the x-sso-date header
2. Extract the x-ayla-origin-host header
3. Extract the Authorization header
4. Extract signed headers list from the Authorization header

---

## Step 3: Verify time

1. Extract the timestamp provided in the x-sso-date header.
2. Verify if the server time in UTC is within 15 seconds of the timestamp provided in this header.

If the request is over 15 seconds old, reject the request by returning a 401 HTTP status code.

---

## Step 4: Calculate signature and Authorization Header

1. Perform the steps listed in [Client Logic](#) to calculate the signature of the request on the server.

There is a step to determine the signing key, used to derive the actual signature. The canonical header can be derived from the list of signed headers provided in the Authorization header.

2. Derive the authorization header.

---

### Step 5: Process the Request

1. Compare the derived authorization header with the header provided in the request.
2. If they match, the client request is valid. Proceed to processing the request.
3. Else, return a 401 HTTP status code to the client.

## 3.2 Ayla Authentication

---

This section specifies the method by which calls to the Ayla service is authenticated.

The provider obtains an SSO Ayla app id and an SSO Ayla app secret from the Ayla Service. This is different from the app id and app secret obtained by typical mobile applications.

Requests to the Ayla service, initiated from the identity provider, should include an Authorization header to allow the Ayla service to authenticate the request.

The request-signing algorithm is identical to the algorithm specified in [Provider Authentication](#), except that roles are reversed. Here, the Ayla service is the server and the Identity provider is the client.

Use the following parameters to compute the signature:

Scope: "user/sso/v1"

Salt: "AYLA-SSO"

## 3.3 API Specification

---

This section specifies the Ayla callback APIs and message formats for information exchange between the Identity Provider service and Ayla service.

*Protocol:* HTTPs

*Message Format:* JSON

*Authentication:* See [Provider Authentication](#)

*Content-Type:* application/json

### 3.3.1 Token Validation API

This API is used by the Ayla service to authenticate a user with a provider-issued token.

*Initiator:* Ayla Service

*Receiver:* Identity Provider

*URI:* Configurable. See [Setup](#). Defaults to "/api/v1/authenticate"



HTTP Method: GET

### Input Parameters:

Name	Description	Type	Mandatory
token	Access token issued by the identity provider to the app.	ASCII String (255 character limit)	Yes
context parameters	Any additional context that the identity provider needs to complete the call. This parameter can be configured on the Ayla service as key-value pairs	Up to 5 parameters are allowed. Name and value are UTF-8 strings, each not exceeding 255 characters.	No

### Response:

HTTP Status Code 200

(if authentication succeeds)

HTTP Status Code 401

(if authentication fails)

### Response Parameters:

Encapsulated in a "response" object (if authentication succeeds)

Name	Description	Type	Mandatory
status	Status code. '1' indicates success. Any other integer code may be included for debugging purposes.	Zero or positive integer	Yes
message	String for debugging purposes. It may be relayed to the app.	UTF-8 String (255 character limit)	Yes
user	User Profile. See <a href="#">User Profile Format</a>	User Object in JSON	Yes, if token is valid

### Sample Response:

```
{
  "response": {
    "status": 1,
    "message": "token valid",
    "user": {
      "uuid": "xxxx",
```

```

        "email": "xxxxx",
        "phone": "xxxxx",
        "firstname": "xxxxxxx",
        "lastname": "xxxxx",
        "nickname": "xxxxxx"
    }
}

```

### 3.3.2 User Profile Format

Mandatory fields are indicated in the table below. More fields may be included.

Name	Description	Type	Mandatory
uuid	Immutable, unique identifier for the user.	UUID, up to 36 characters in length	Yes
email	Email of the user	ASCII string, up to 254 characters	Yes
phone	Phone number of the user	String, up to 16 characters	No
firstname	First Name of the user	UTF-8 String, up to 255 characters long	Yes
lastname	Last Name of the user	UTF-8 String, up to 255 characters long	Yes
nickname	Nickname of the user	UTF-8 String, up to 255 characters long	No

#### Sample

```

{
  "user": {
    "uuid": "xxxx",
    "email": "xxxxx",
    "phone": "xxxxx",
    "firstname": "xxxxxxx",
    "lastname": "xxxxx",
    "nickname": "xxxxxx"
  }
}

```

```
}

```

### 3.3.3 Get User Profile API

The Ayla service uses this API to get a user profile with a UUID, specified as a URL parameter.

*Initiator:* Ayla Service

*Receiver:* Identity Provider

*URI:* Configurable. See [Setup](#). Defaults to "/api/v1/userprofile"

*HTTP Method:* GET

Authentication: See [Provider Authentication](#)

#### Input Parameters:

Name	Description	Type	Mandatory
uuid	The access token issued by the identity provider to the app.	UUID String, upto 36 characters in length	Yes

#### Response:

HTTP Status Code 200

(if authentication succeeds)

HTTP Status Code 401

(if authentication fails)

#### Response Parameters:

Encapsulated in a "response" object (if authentication succeeds)

Name	Description	Type	Mandatory
status	Status code '0' indicates success. Status code '1' indicates "user not found". Any other integer code may be included for debugging purposes.	Zero or positive integer	Yes
message	String for debugging purposes. It may be relayed to the app.	UTF-8 String (255 character limit)	Yes
user	User Profile. See <a href="#">User Profile Format</a>	User Object in JSON	Yes, if token is valid

**Sample Success Response:**

```
{
  "response": {
    "status": 0,
    "message": "valid user",
    "user": {
      "uuid": "xxxx",
      "email": "xxxxxx",
      "phone": "xxxxxx",
      "firstname": "xxxxxx",
      "lastname": "xxxxxx",
      "nickname": "xxxxxx"
    }
  }
}
```

**Sample Failure Response (If user does not exist):**

```
{
  "response": {
    "status": 1,
    "message": "Invalid user",
  }
}
```

### 3.3.4 Update and Delete Accounts

This API is called by the provider when there is an action on the user account.

*Initiator:* Identity Provider

*Receiver:* Ayla Service

*URI:* /api/v1/ssouser

*HTTP Method:* PUT

Authentication: See [Ayla Authentication](#)

**Input Parameters:**

Name	Description	Type	Mandatory
operation	Indicates what the operation is on the user	ASCII String	Yes

	account	Valid Values:  "UPDATE", "DELETE"	
uuid	UUID of the user account for which the action occurred	UUID	Yes

### Response:

HTTP Status Code 202

(if Update request is Accepted)

HTTP Status Code 401

(if authentication fails)

HTTP Status Code 403

(if authentication fails)

=====

### Example 1:

INPUTS

algorithm="HMAC-SHA256"

url = "https://user.aylanetworks.com/api/v1/ssouser"

scope="user/sso/v1"

salt = "AYLA-SSO"

x-ayla-origin-host="user.aylanetworks.com"

time = "20151123T224515Z"

request\_params = {"operation" => "DELETE", "uuid" => "e4194664-9233-11e5-ac92-065eed1a9f3b"}

request\_method = "PUT"

provider\_app\_id="acme-sso-id"

provider\_app\_secret = "acme-secret"

app\_id = "ACMEDev-id"

app\_secret = "ACMEDev-5991211"

### COMPUTED VALUES:

signed\_headers : x-ayla-origin-host;x-sso-date

canonical\_headers : x-ayla-origin-host: user.aylanetworks.com\nx-sso-date: 20151123T224515Z\n

canonical\_uri : /api/v1/ssouser

```
canonical_query_string : operation=DELETE&uuid=e4194664-9233-11e5-ac92-065eed1a9f3b
```

```
STRING TO SIGN :
```

```
HMAC-SHA256
```

```
20151123T224515Z
```

```
user/sso/v1
```

```
PUT
```

```
/api/v1/ssouser
```

```
operation=DELETE&uuid=e4194664-9233-11e5-ac92-065eed1a9f3b
```

```
x-ayla-origin-host: user.aylanetworks.com
```

```
x-sso-date: 20151123T224515Z
```

```
x-ayla-origin-host;x-sso-date
```

```
SIGNING KEY :
```

```
Q\xB8R}\xF9O\xC1\xB00A'6\xC4\xBF\xCD\xAFn\xE0\xBB)\xB4\xD67 \x86f\x81\ee\xCF\x7F\xF8
```

```
AUTH HEADER :
```

```
HMAC-SHA256 Credential=ACMEDev-id/user/sso/v1, SignedHeaders=x-ayla-origin-host;x-sso-date, Signature=5b3b428f46466fe0e76cc71fb6dd3ad30df68c20153fba59cfc3e5e711cf65de
```

```
=====
```

### 3.3.5 Login with Identity Provider Access Token

This API is called by the app to login to the Ayla service with the Identity Provider issued access token.

*Initiator:* Mobile application/Other end user application

*Receiver:* Ayla Service

*URI:* /api/v1/token\_sign\_in

*HTTP Method:* POST

*Authentication:* n/a

### Input Parameters:

Name	Description	Type	Mandatory
app_id	The application id that the end user app. obtains from the Ayla service	Alphanumeric string issued by Ayla	Yes
app_secret	The application secret that the end user app. obtains from the Ayla service	Alphanumeric string issued by Ayla	Yes
token	Identity Provider issued token, obtained during the initial phase of log-in	Alphanumeric string issued by the identity provider. Maximum length: 255 characters	Yes

### Response:

HTTP Status Code 200

(if authentication succeeds)

HTTP Status Code 401

(if authentication fails)

### Response Parameters (if authentication succeeds):

Name	Description	Type	Mandatory
access-token	The Ayla auth token that can be used to make further calls to the Ayla service	Alphanumeric string issued by Ayla. Maximum length: 255 characters	Yes
refresh-token	The Ayla refresh token that can be used to obtain a new Ayla auth token after the Ayla auth token expires.	Alphanumeric string issued by Ayla. Maximum length: 255 characters	Yes
expires-in	The number of seconds after which it expires	integer	Yes
role	The role that the user has. Indicates the privilege level of the token.	Alphanumeric string issued by Ayla	Yes

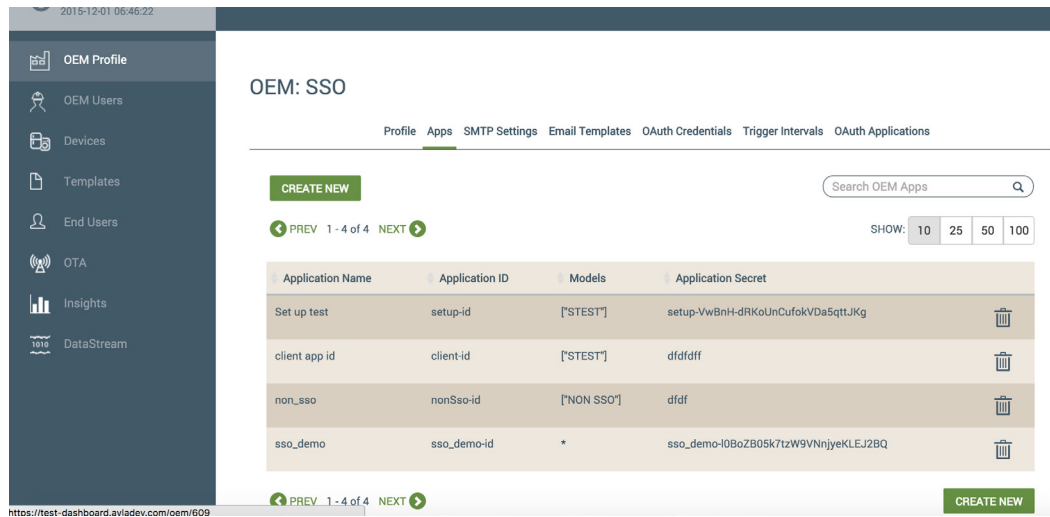
### Sample Success Response:

```
{ "access_token": "aea5f45e3bfb45fa8715b6bc2e1039dc", "refresh_token": "192fd8792ae14fcfa1dec3940b57dd00", "expires_in": 86400, "role": "EndUser" }
```

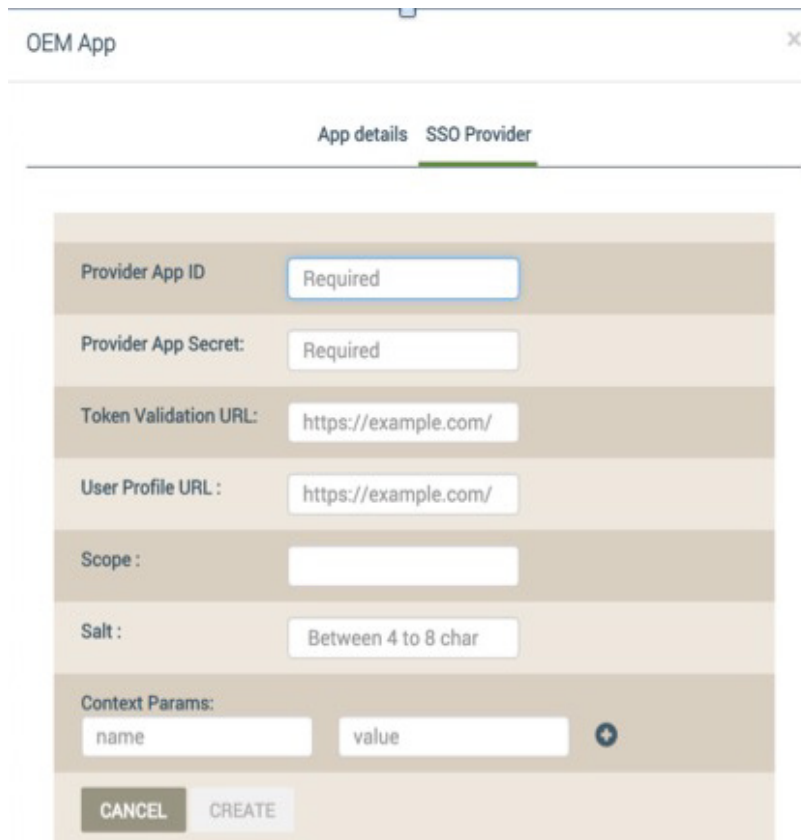
### 3.3.6 Add Your IDP

Create your IDP. An example with working code of Python Based Flask IDP server shows how to link your IDP (Identity Provider) with Ayla Services.

1. On the OEM Dashboard Navigation Menu: OEM Profile > Apps tab, select the App you want to add.



2. On the 'SSO Provider' tab, enter the following parameters:



**OEM App**

App details SSO Provider

Provider App ID: Required

Provider App Secret: Required

Token Validation URL: https://example.com/

User Profile URL: https://example.com/

Scope:

Salt: Between 4 to 8 char

Context Params:

name	value

CANCEL CREATE

- Provider App ID (same used to calculate auth on Provider side)



- Provider App Secret (same used to calculate auth on Provider side)
- Token Validation URL (URL used by Ayla service to validate user tokens)
- User Profile URL (URL used by Ayla service to obtain user profiles)
- Scope (same used to calculate auth on Provider side)
- Salt (same used to calculate auth on Provider side)
- (Optional) Context params *if any*.

OEM App ×

App details SSO Provider

Provider App ID

sso\_demo\_id

Provider App Secret:

sso\_demo\_secret

Token Validation URL:

https://idp.com/api/v1/

User Profile URL :

https://idp.com/api/v1/

Scope :

scope

Salt :

salt

Context Params:

name

value

+

CANCEL

CREATE

3. Click **CREATE**.

4. Verify the information.

OEM App

App details

SSO Provider

Provider App ID:	sso_demo_id
Provider App Secret:	sso_demo_secret
Token Validation URL:	https://idp.com/api/v1/authenticate
User Profile URL:	https://idp.com/api/v1/user
Scope :	scope
Salt :	salt
Context Params	
Name	Value

CLOSE

EDIT

DELETE

5. Close the dialog box. The provider is ready to go!

## 4 Curl Examples & Explanation

### 4.1 User Creation at Initial Auth

User created on Ayla during first auth. The following example is from the client (e.g. mobile) to Ayla's User service to sign in.

**Request:**

```
curl 'https://user.aylanetworks.com/api/v1/token_sign_in' -H 'Content-Type: application/json' -d '{"token": "abcf1f722950d84c44ab21e9976d462e169b58d", "app_id": "test-app", "app_secret": "test-3431389"}'
```

**Response:**

Code: 200

```
{
  "access_token": "a0e03d8af0524d0e9f6f0328167aa4fb",
  "refresh_token": "d7c155b7d3f84cfca5fca5f76167d503",
  "expires_in": 86400,
  "role": "EndUser",
  "role_tags": [
  ],
  "code": "ok"
}
```

**Error Code:**

406 – missing -H 'Accept: application/json'

### 4.2 Token validate

From Ayla to the IDP to validate and return the user profile in the response upon success.

**Request:**

```
curl 'https://idp-emulation-url/api/v1/authenticate?token=<token>&<context_param1_name>=<context_param1_value>' -H 'Authorization HMAC-SHA256 Credential=ilya-app/user/sso/v1, SignedHeaders=x-ayla-origin-host;x-sso-date, Signature=2926bf66da2bcc889d56efdgdfg02641708247b4ffa2f49eb5ac753b4a68f2999'
```

```
' -H 'X-SSO-DATE: 20151123T224515Z'
\
```

**Response:**

Code: 200

```
{
  "response": {
    "status": 1,
    "message": "token valid",
    "user": {
      "uuid": "",
      "email": "",
      "phone": "",
      "firstname": "",
      "lastname": "",
      "nickname": ""
    }
  }
}
```

## 4.3 Get User Profile

Ayla calls this API on IDP when Update user call was performed.

**Request:**

```
curl 'https://idp-emulation-url/api/v1/authenticate?uuid=<uuid>' -H 'Authorization
HMAC-SHA256 Credential=ilya-app/user/sso/v1, SignedHeaders=x-ayla-origin-host;x-sso-
date, Signature=2926bf66da2bcc889d56efdgdfg02641708247b4ffa2f49eb5ac753b4a68f2999'
' -H 'X-SSO-DATE: 20151123T224515Z'
```

**Response:**

Code: 200

```
{
  "response": {
    "status": 1,
    "message": "valid user",
    "user": {
      "uuid": "",
```

```

        "email": "",
        "phone": "",
        "firstname": "",
        "lastname": "",
        "nickname": ""
    }
}
}

```

## 4.4 Update User

This API called by IDP to Ayla when user is updated.

### Request:

```

curl -X PUT -H "Content-Type: application/json" -H "Authorization: HMAC-SHA256
Credential=ilya-app/user/sso/v1, SignedHeaders= x-ayla-origin-host;x-sso-date,
Signature=2926bf66da2bcc889d56efdgdfhgg02641708247b4ffa2f49eb5ac753b4a68f2999" -H "X-
SSO-DATE: 20151123T224515Z" -d '{
    "operation": "UPDATE",
    "uuid": "e4194664-9233- 11e5-ac92-065eed1a9f3b"
}' 'https://user.aylanetworks.com/api/v1/ssouser'

```

### Response

202

## 4.5 Delete User

This API called by IDP to Ayla when user is deleted.

### Request:

```

curl -X PUT -H "Content-Type: application/json" -H "Authorization: HMAC-SHA256
Credential=ilya-app/user/sso/v1, SignedHeaders= x-ayla-ori
gin-host;x-sso-date,
Signature=2926bf66da2bcc889d56efdgdfhgg02641708247b4ffa2f49eb5ac753b4a68f2999" -H "X-
SSO-DATE: 20151123T224515Z" -d '{
    "operation": "DELETE",
    "uuid": "e4194664-9233- 11e5-ac92-065eed1a9f3b"
}' 'https://user.aylanetworks.com/api/v1/ssouser'

```

### Response:

202

\*All Authorization headers just an examples and not real working headers.

## 4.6 Timeout

---

All calls to the identity provider must return in 15 seconds. The call fails if a response is not received within this time. No retries are made.

## 5 Setup

An external identity provider can be configured on the Ayla service with these parameters.

The OEM dashboard will be enhanced to provide a UI for this purpose.

- **Provider App Id**  
Identifies Ayla as a client with the identity provider.
- **Provider App Secret**  
Derives an HMAC signature. See [Provider Authentication](#)  
Ayla recommends the provider app secret be at least 256 bits, generated with a secure random key generator.
- **Token Validation URL**  
This URL is invoked to validate the provider-issued token. See [Token Validation API](#). The domain name of the URL entered here should be the same as the domain name in the OEM profile.
- **Context Parameters**  
Up to five name-value pairs passed as URL parameters in the [Token Validation API](#)
- **Scope**  
A string used in the HMAC authentication signature. Can be used as metadata on the provider.
- **Salt**  
A string added to the provider app secret to derive the signing key.

### 5.1 Client Signing Sample Code (Ruby)

```
#!/usr/bin/env ruby
require 'open-uri'
require 'openssl'
require 'time'
require 'base64'

#CONSTANTS
algorithm="HMAC-SHA256"
```

```
#
#PARAMETERS CONFIGURED BY THE IDENTITY PROVIDER
#
scope="user/sso/v1"
provider_app_id="provider-id"
provider_app_secret = "FwUPD7+ol9b54CXk/OCL1U8m+qXc7ivbnCVzJJxw"
salt = "AYLA-SSO"

#
#PARAMETERS SENT IN THE REQUEST
#
uri="/userinfo"
host="idp.provider.com"
query_string="token=9b54CXk/OCL1U8m+qXc"

#UTILITY METHOD
def sign (signing_key, string_to_sign, hexencode=true)
  digest = OpenSSL::Digest::SHA256.new
  if hexencode
    signature = OpenSSL::HMAC.hexdigest(digest, signing_key,
string_to_sign.force_encoding("utf-8"))
  else
    signature = OpenSSL::HMAC.digest(digest, signing_key,
string_to_sign.force_encoding("utf-8"))
  end
  signature
end

#UTILITY METHOD
def determine_signing_key (salt,provider_app_secret,timestamp)
  key_timestamp = sign((provider_app_secret+salt).encode('utf-8'), timestamp, false)
  key_timestamp
end

#
```



```
#STEP 1: CREATE A CANONICAL REQUEST
#

#Define the HTTP verb
request_method="GET"

#Determine the UTC time in ISO8601 format
#time=Time.now.utc.strftime("%Y%m%dT%H%M%S") # CORRECT VERSION
time="20150817T063855Z" # HARD CODE TIME FOR TESTING PURPOSES

#Define the canonical URI
canonical_uri="/userinfo"

#Create the canonical query string
query_string="token=9b54CXk/OCL1U8m+qXc&context=somecontext"
token_value=URI::encode("9b54CXk/OCL1U8m+qXc")
canonical_query_string="context=somecontext&token=" + token_value

#Create the list of signed headers
signed_headers="x-ayla-origin-host;x-sso-date"

#Create the canonical header
canonical_headers= "x-ayla-origin-host: " + host + "\n" + "x-sso-date: " + time + "\n"

#Create the canonical request
canonical_request=request_method+"\n"+canonical_uri+"\n"+canonical_query_string+"\n"+
canonical_headers+"\n"+signed_headers

puts "Canonical Request: #{Base64.encode64(canonical_request)}" # Base64 encode for
readability

#

#STEP 2: CREATE A STRING TO SIGN
#

#Define the scope string
# - Specified as part of identity provider config
```

```
#Determine the signing algorithm
# - This is pre-defined

#Create the complete string to sign
string_to_sign=algorithm+"\n"+time+"\n"+scope+"\n"+canonical_request
puts "String to sign: #{Base64.encode64(string_to_sign)}"

#
#Step 3: CREATE A SIGNING KEY
#
signing_key = determine_signing_key(salt,provider_app_secret,time)
puts "Signing key: #{Base64.encode64(signing_key)}"

#
#Step 4: CREATE A SIGNATURE
#
signature = sign(signing_key,string_to_sign)

#
#Step 5: CREATE AN AUTHORIZATION HEADER
#
authorization_header="Authorization: " + algorithm + " " + "Credential=" +
provider_app_id + "/" + scope + ", " + "SignedHeaders=" + signed_headers + ", " +
"Signature=" + signature
puts "authorization_header: #{authorization_header}"

#
#STEP 6: MAKE THE REQUEST TO THE PROVIDER
#
#Request should include x-ayla-origin-host header, date header, authorization header
```

## 5.2 Provider Verification Sample Code (Ruby)

```
#!/usr/bin/env ruby
require 'open-uri'
```

```
require 'openssl'
require 'time'
require 'base64'

#CONSTANTS
algorithm="HMAC-SHA256"

#
#PARAMETERS CONFIGURED BY THE IDENTITY PROVIDER
#
scope="user/sso/v1"
provider_app_id="provider-id"
provider_app_secret = "FwUPD7+ol9b54CXk/OCL1U8m+qXc7ivbnCVzJJxw"
salt = "AYLA-SSO" #Provider determines the salt

#
#PARAMETERS DERIVED FROM THE CLIENT REQUEST
#
uri="/userinfo"
host_header_in_request = "x-ayla-origin-host: idp.provider.com"
date_header_in_request = "x-sso-date: " + "20150817T063855Z"

#UTILITY METHOD
def sign (signing_key, string_to_sign, hexencode=true)
  digest = OpenSSL::Digest::SHA256.new
  if hexencode
    signature = OpenSSL::HMAC.hexdigest(digest, signing_key,
string_to_sign.force_encoding("utf-8"))
  else
    signature = OpenSSL::HMAC.digest(digest, signing_key,
string_to_sign.force_encoding("utf-8"))
  end
  signature
end
```

```
#UTILITY METHODkey_timestamp = sign((provider_app_secret+salt).encode('utf-8'),
timestamp, false)

key_timestamp = sign((salt+provider_app_secret).encode('utf-8'), timestamp, false)
    key_timestamp
end

#
#STEP 1: DETERMINE PROVIDER PARAMETERS
#
# Determine scope and salt from the parameters configured by the provider
# Determine the URI from the request

request_method="GET"
server_time = Time.now.utc.to_time.iso8601

#
#STEP 2: EXTRACT REQUEST PARAMETERS
#
host="idp.provider.com"
time_in_date_header = "20150817T063855Z"
query_string="token=9b54CXk/OCL1U8m+qXc&context=somecontext"

#
#STEP 3: VERIFY TIME
#
# Verify the the timestamp in the "x-sso-date" header is within 15 seconds of the
server time

#
#Step 4: Calculate signature and Authorization Header
#

#Define the canonical URI
```

```
canonical_uri="/userinfo"

#Create the canonical query string
token_value=URI::encode("9b54CXk/OCL1U8m+qXc")
canonical_query_string="context=somecontext&token=" + token_value

#Create the list of signed headers
signed_headers="x-ayla-origin-host;x-sso-date"

#Create the canonical header
# Split signed_headers by ';'. For each header, extract header from request and
construct canonical header.
# Omitted for brevity
canonical_headers=host_header_in_request + "\n" + date_header_in_request + "\n"

#Create the canonical request
canonical_request=request_method+"\n"+canonical_uri+"\n"+canonical_query_string+"\n"+c
anonical_headers+"\n"+signed_headers
puts "Canonical Request: #{Base64.encode64(canonical_request)}"

#Create the complete string to sign
string_to_sign=algorithm+"\n"+time_in_date_header+"\n"+scope+"\n"+canonical_request
puts "String to sign: #{Base64.encode64(string_to_sign)}"

#Create the signing key
signing_key = determine_signing_key(salt,provider_app_secret,time_in_date_header)
puts "Signing key: #{Base64.encode64(signing_key)}"

#Create the signature
signature = sign(signing_key,string_to_sign)

#Create the authorization header
authorization_header="Authorization: " + algorithm + " " + "Credential=" +
provider_app_id + "/" + scope + ", " + "SignedHeaders=" + signed_headers + ", " +
"Signature=" + signature
puts "authorization_header: #{authorization_header}"
```

```
# Request should include x-ayla-origin-host header, date header, authorization header
# Compare authorization header in request to computed authorization header
# If the computed authorization header matches the authorization header in the
request, the #request is authenticated. Process the request
```

## 6 Troubleshooting

**Your IDP gets 401 when it tries to DELETE or UPDATE the user on Ayla.**

Most likely the signature is different from what Ayla has calculated. To resolve, make sure of the following conditions:

1. Send 'operation=delete' / 'operation=update' and 'uuid=<uuid>' in the url as url params – and no typos in the strings.
2. Send 'x-sso-date' header with the correct value of a date in this format: 20151123T224515Z.
3. Use 'PUT' method to calculate a signature.
4. Use the right URI to calculate a signature ('/api/v1/ssouser').
5. Use the right ayla x-ayla-origin-host to calculate a signature.
6. Use correct 'salt' and 'scope'(same which you used to create IDP entry) to calculate the signature.
7. Use correct app\_id and app\_secret to calculate the signature.
8. Put log/prints into IDP code to compare values intended for use with what was used.
9. Use the correct algorithm. Here is working Python example of all steps in one function.

```
import hashlib
import hmac
```

10. Put log/print on each step of calculation and do manual calculation with the command line to compare each value.
11. If nothing has helped, contact Ayla Support and be ready to provide logs of your IDP or direct access to the machine.

**Ayla sends your IDP the wrong signature, IDP responds with 401**

Complete all of the steps from the previous case. The calculation process is the same.

In this case, values (host, uri, method and url params) are different, as shown below:

```
@staticmethod
def calculate_authorization(method, uri, host, app_id, app_secret, utc, salt, scope, params):
    canonical_query_string = ''
    for key, value in sorted(params.items()):
        canonical_query_string += '%s=%s&' % (key, value)
    canonical_query_string = canonical_query_string[:-1]
    signed_headers = '{0};{1}'.format(AYLA_HOST_HEADER, AYLA_DATE_HEADER)
    canonical_headers = '{0}: {1}\n{2}: {3}\n'.format(AYLA_HOST_HEADER, host, AYLA_DATE_HEADER, utc)
    signing_algorithm = 'HMAC-SHA256'
    complete_string_to_sign = signing_algorithm + '\n' + utc + '\n' + scope + '\n' + method + '\n' + uri + '\n' + \
        canonical_query_string + '\n' + canonical_headers + '\n' + signed_headers
    print(complete_string_to_sign)
    s = hmac.new(app_secret + salt, utc, digestmod=hashlib.sha256).digest()
    print(repr(s))
    signature = hmac.new(s, complete_string_to_sign, digestmod=hashlib.sha256).hexdigest()
    header = signing_algorithm + ' ' + 'Credential=' + app_id + '/' + scope + ', ' + 'SignedHeaders=' + \
        signed_headers + ', ' + 'Signature=' + signature
    return header
```

```
AYLA_HOST_HEADER = 'x-ayla-origin-host'
AYLA_DATE_HEADER = 'x-sso-date'
```

## 6.1 Example 1

### INPUTS:

```
algorithm="HMAC-SHA256"
url = "https://user.aylanetworks.com/api/v1/ssouser"
scope="user/sso/v1"
salt = "AYLA-SSO"
x-ayla-origin-host="user.aylanetworks.com"
time = "20151123T224515Z"
request_params = {"operation" => "DELETE", "uuid" => "e4194664-9233-11e5-ac92-065eed1a9f3b"}
request_method = "PUT"
provider_app_id="acme-sso-id"
provider_app_secret = "acme-secret"
app_id = "ACMEDev-id"
app_secret = "ACMEDev-5991211"
```

### COMPUTED VALUES:

```
signed_headers : x-ayla-origin-host;x-sso-date
canonical_headers : x-ayla-origin-host: user.aylanetworks.com\nx-sso-date:
20151123T224515Z\n
canonical_uri : /api/v1/ssouser
canonical_query_string : operation=DELETE&uuid=e4194664-9233-11e5-ac92-065eed1a9f3b
```



**STRING TO SIGN:**

```
HMAC-SHA256
20151123T224515Z
user/sso/v1
PUT
/api/v1/ssouser
operation=DELETE&uuid=e4194664-9233-11e5-ac92-065eed1a9f3b
x-ayla-origin-host: user.aylanetworks.com
x-sso-date: 20151123T224515Z
```

```
x-ayla-origin-host;x-sso-date
```

**SIGNING KEY:**

```
Q\xB8R}\xF90\xC1\xB00A'6\xC4\xBF\xCD\xAFn\xE0\xBB)\xB4\xD67 \x86f\x81\ee\xCF\x7F\xF8
```

**AUTH HEADER:**

```
HMAC-SHA256 Credential=ACMEDev-id/user/sso/v1, SignedHeaders=x-ayla-origin-host;x-sso-date, Signature=5b3b428f46466fe0e76cc71fb6dd3ad30df68c20153fba59cfc3e5e711cf65de
```





4250 Burton Drive, Santa Clara, CA 95054

Phone: +1 408 830 9844

Fax: +1 408 716 2621