# Identity Management API Guide

*Updated May 8, 2020*

Use this guide to work with the SecureAuth IdP Identity Management APIs to leverage end user profiles from the configured data store, add and update profiles, and modify attributes in profiles. These tools enable administrators to manage end users programmatically from the website, without building connections directly to the data store. A use case example is where an administrator uses the Identity Management APIs to access the data store for a self-service portal where end users can reset forgotten passwords, change expired passwords, and change knowledge-based answers and PINs.

Use API Identity Management tools with Authentication API features configured on the SecureAuth IdP realm to securely direct end users through unique logins and interfaces without leaving the application.

---

## Prerequisites

1. Have access to the application code.
2. Have an on-premises Active Directory (AD) with which SecureAuth IdP can integrate.

   > The Multi-Data Store option is not compatible with the Identity Management APIs.

3. Create a new realm or access an existing realm where the Identity Management APIs are enabled.

   You can include the API in any realm with any post authentication event as long as you integrate the appropriate directory and configure the appropriate settings.
4. Configure the **Data** tab in the SecureAuth IdP Web Admin.

   The **GET /user** and **update /user** functions can work with non-AD integrations.

---

## Workflow

The following workflow guides you through the different SecureAuth IdP pieces necessary to configure the Identity Management APIs. Use the steps as a check list or move through the sections that follow in sequential order.

1. Complete the SecureAuth IdP Web Admin configuration steps setup.
2. Configure the Authorization Header & Response Header for all GET and POST requests.
3. Configure /users endpoints and configure group association endpoints for users and groups.

---

## SecureAuth IdP Web Admin configuration steps

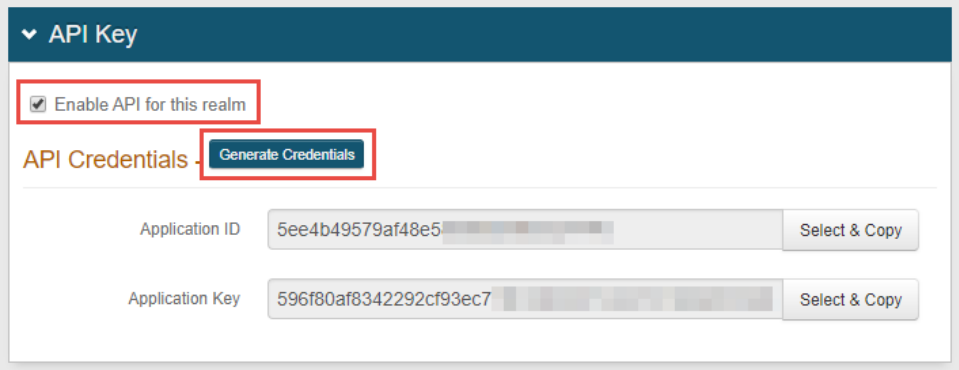
Only API steps are required; all other Web Admin steps are optional and should be performed based on the features you want to implement.

Make the changes to the following sections in the appropriate SecureAuth IdP realm **API** tab.

### API Key section

1. In the **API** tab, go to the **API Key** section.

2. Check **Enable API for this realm**.



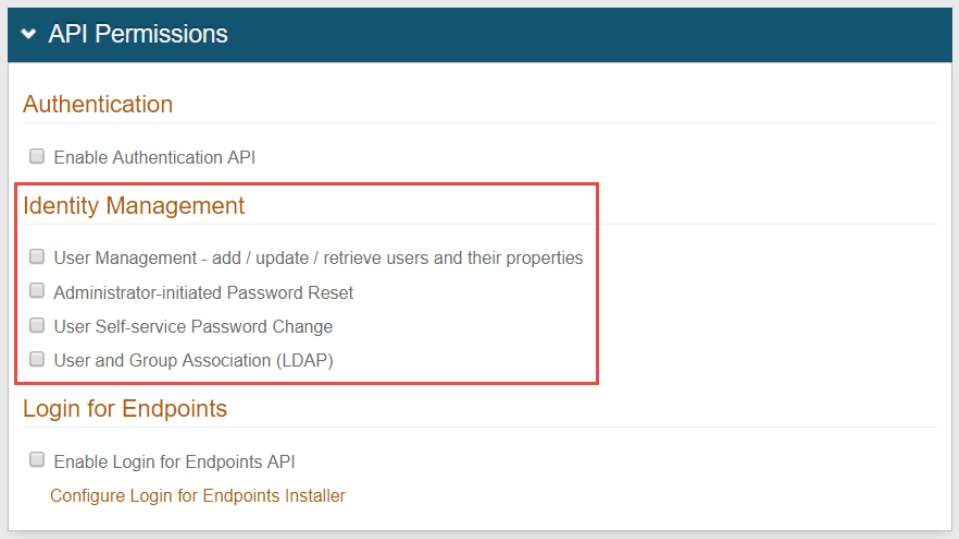

3. Click **Generate Credentials** to create a new **Application ID** and **Application Key.**

   The **Application ID** and **Application Key** are unique for each realm.

   The API key looks like it comprises **64** random characters, but it actually comprises **32** two-character base-16 hexadecimal values. This is important when using the API key to produce the HMAC hash. These values are required in the Header configuration steps you will perform later.

   This is important to note when using the API key to produce the HMAC hash.
4. Click **Select & Copy** to copy the contents from the fields.

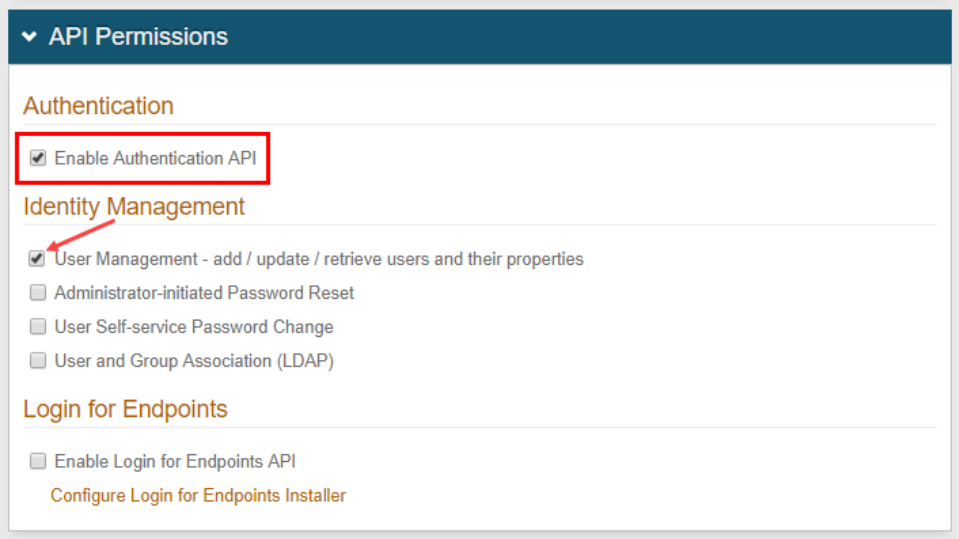
   These values are required in the Header configuration steps you will perform later.
5. In the **API Permissions** section, check **Enable Authentication API.**




6. Check at least one **Identity Management tool to include in the API.**

   **User management - add / update / retrieve users and their properties**: Use this tool to add new user profiles, and to retrieve and update existing user profiles. Updating a user profile includes setting and clearing property values in the user profile.

   If your team is using SecureAuth RADIUS 2.4.15 or later, you must check **User Management**. This setting enables the SecureAuth IdP API to connect to User properties.




**Administrator initiated password reset**: Use this tool to let an administrator send the end user a new password requested by using an application.

Use case: End users request a new password because they have forgotten the current one.

**User self-service password change**: Use this tool to let the end user input both the current password and a new password

Use case: With the **Administrator initiated password reset** option, end users enter the password sent by the administrator (the current password), and then enter a new password to perform the reset.

**User & group association (LDAP)**: Use this tool to enable associations between existing users and groups within the LDAP data store.

7. Save the configuration.

---

# Configure request header

Authentication against an API requires a configured HTTP basic authorization header and Content-Type header.

1. Add a Content-Type header with a value of **application/json**

2. Create an Authorization header for GET and POST requests using the steps below.

## Authorization request header

A. Build a string based on the request:

A1. **METHOD (GET)** or **METHOD (POST)**

A2. **DATE/TIME**

| Header types | IdP version | String requirements |
|---|---|---|
| Date | v9.1+ | second-precision timestamp |
| X-SA-Date (custom) | v9.1+ | second-precision timestamp |
| X-SA-Ext-Date (custom) | v9.2+ | millisecond-precision timestamp |

A3. **APPLICATION ID** (from the Identity Platform Web Admin – API Key section).

A4. **PATH** (API endpoint). For example: /secureauth2/api/v3/users/{userID}

B. Create an HMAC SHA256 hash of step 3 using the **Application Key** (from the Identity Platform Web Admin – API Key section).

This step is executed by calling the HMAC and producing the hash value.

C. Encode the HMAC SHA256 hash from step 3 in Base64.

D. Concatenate the "**Application ID**", "**:**", and the "**Base64 encoded HMAC SHA256 hash**" from step 3:

```
ApplicationID:Base64EncodedHMACSHA256Hash
```

E. Encode the value from step 4 in Base64.

F. Concatenate "**Basic**" and the "**Value of Step 5**":

```
Basic Step5Value
```

### Authorization header GET / POST request examples

## GET request example

```
Step A
    GET
    Wed, 08 Apr 2015 21:37:33.123 GMT
    1b700d2e7b7b4abfa1950c865e23e81a
    /secureauth2/api/v1/users/jsmith/factors

End Result: "GET\nWed, 08 Apr 2015 21:37:33.123 GMT\n1b700d2e7b7b4abfa1950c865e23e81a\n/secureauth2/api/v1/users
/jsmith/factors"

Step B
        Non-printable bytes are produced in this step

Step C
    F5yqdLDJddUYOlrpBlOJBh/YCUIMVCsWejuhiCrqMmw=

Step D
    1b700d2e7b7b4abfa1950c865e23e81a:F5yqdLDJddUYOlrpBlOJBh/YCUIMVCsWejuhiCrqMmw=

Step E
    MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOnorVGNYNG4vbFlsTmNvNjRpQkRENVJKaHFiZ0h0UGYwaEQ4d1d4bTgvWVk9

Step F
    Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOnorVGNYNG4vbFlsTmNvNjRpQkRENVJKaHFiZ0h0UGYwaEQ4d1d4bTgvWVk9

End Result:
    Method: GET,
    RequestUri: 'https://secureauth.company.com/secureauth2/api/v1/users/jsmith/factors',
    Version: 1.1,
    Headers: {
        Connection: Keep-Alive
        X-SA-Ext-Date: Wed, 08 Apr 2015 21:37:33.123 GMT
        Authorization: Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkY1eXFkTERKZGRVWU9scnBCbE9KQmgvWUNVSU1WQ3NXZWp1aGlDcnFNbXc9
        Host: secureauth.company.com
        Content-Length: 0
    }
```

## POST request example

```
Step A
    POST
    Wed, 08 Apr 2015 21:27:30.123 GMT
    1b700d2e7b7b4abfa1950c865e23e81a
    /secureauth2/api/v1/auth
    {"user_id":"jsmith","type":"user_id"}

End Result: "POST\nWed, 08 Apr 2015 21:27:30.123 GMT\n1b700d2e7b7b4abfa1950c865e23e81a\n/secureauth2/api/v1
/auth\n{"user_id":"jsmith","type":"user_id"}"

Step B
        Non-printable bytes are produced in this step

Step C
```

```
    D6nkepAEtk/M+cpkyWQ/hZMXZxPJ32L++5ZZa6+pB8U=

Step D
    1b700d2e7b7b4abfa1950c865e23e81a:D6nkepAEtk/M+cpkyWQ/hZMXZxPJ32L++5ZZa6+pB8U=

Step E
    MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

Step F
    Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

End Result:
    Method: POST
    RequestUri: 'https://secureauth.company.com/secureauth2/api/v1/auth'
    Version: 1.1
    Headers: {
        Connection: Keep-Alive
        X-SA-Ext-Date: Wed, 08 Apr 2015 21:27:30.123 GMT
        Authorization: Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9
        Expect: 100-continue
        Host: secureauth.company.com
        Content-Length: 36
        Content-Type: application/json; charset=utf-8
    }
```

### Authorization header non-validation response examples

## Authorization Header non-validation responses

When an Authorization header cannot be validated, one of the following responses is returned:

| | |
|---|---|
| `{`<br>`  "status": "invalid",`<br>`  "message": "Missing authentication header.",`<br>`}` | `{`<br>`  "status": "invalid",`<br>`  "message": "Unknown authentication scheme.",`<br>`}` |
| `{`<br>`  "status": "invalid",`<br>`  "message": "Clock skew of message is outside threshold.",`<br>`}` | `{`<br>`  "status": "invalid",`<br>`  "message": "AppId is unknown.",`<br>`}` |
| `{`<br>`  "status": "invalid",`<br>`  "message": "Authentication header value is empty.",`<br>`}` | `{`<br>`  "status": "invalid",`<br>`  "message": "Authentication header has been seen before.",`<br>`}` |
| `{`<br>`  "status": "invalid",`<br>`  "message": "Authentication header value's format should be 'appId:hash'.",`<br>`}` | `{`<br>`  "status": "invalid",`<br>`  "message": "Invalid credentials.",`<br>`}` |

3. OPTIONAL: If using the Email two-factor authentication method and a language other than US English, create an **Accept-Language** header to generate the Email OTP messages in the preferred language.

    If the no **Accept-Language** header is present, then the Email OTP messages default to US English.

# Configure response header

SecureAuth's API includes security hashing that ensures the integrity of the information being sent in all of the endpoints' responses from the appliance to the application.

Through a hashing algorithm, SecureAuth IdP delivers a signature that can be validated by the application to ensure that no data manipulation has occurred prior to the application consuming the data.

Before sending the response to the application (initiated by the endpoint request), SecureAuth IdP creates the signature and includes it in the response header (prepended by X-SA-SIGNATURE:). The application can then validate the response by hashing the date or time and content from the consumed response and the Application ID with the Application Key and compare the new hashed value with the X-SA-SIGNATURE value. The Application ID and Application Key are generated in SecureAuth IdP and connect the appliance with the application for each endpoint transaction.

## Application response header

In the application's code, the following is required to validate the response header's signature.

1. Build a string based on the request.

   a. **X-SA-DATE** for a second-precision timestamp (from the SecureAuth IdP v.1 or later response).

   b. **APPLICATION ID** (from SecureAuth IdP Web Admin – API Key section).

   c. **CONTENT** (JSON Parameters from the SecureAuth IdP response).
2. Create an HMAC SHA256 hash of step 2 using the **Application Key** (from SecureAuth IdP Web Admin – API Key section).

   This step is executed by calling the HMAC and producing the hash value.
3. Encode the HMAC SHA256 hash from step 3 in Base64.
4. Compare the HMAC SHA256 hash from step 3 to the **X-SA-SIGNATURE** value in the SecureAuth response header.
5. Consume the response based on the comparison result.

After hashing the data, if the value exactly matches the signature provided in the SecureAuth IdP response header, then the data has not been compromised. If the value does not match the response signature, then the data has been modified.

---

# Optional: Configure X-SA-Ext-Date header

The string section for **DATE/TIME** can be configured to use either the second-precision UTC time or the millisecond-precision format DateTime.

If using the millisecond-precision, the date string must be included in the X-SA-Ext-Date header.

## X-SA-Ext-Date header example

```
var dateMillis = request.Headers.Date.Value.UtcDateTime.ToString("ddd, dd MMM yyyy HH:mm:ss.fff G\\MT");
                request.Headers.Add("X-SA-Ext-Date", dateMillis);
                request.Headers.Remove("Date");
                var httpMethod = request.Method.Method;
                string uri = request.RequestUri.AbsolutePath;
                string content = null;
                if (request.Content != null)
                {
                    content = request.Content.ReadAsStringAsync().Result;
                }
                result = (string.IsNullOrEmpty(content)) ?
                    string.Join("\n", httpMethod
dateMillis, appId, uri) :
                    string.Join("\n", httpMethod, dateMillis, appId, uri, content);
```

# Configure /users endpoints

The following endpoints are prepended with the URL, https://SecureAuthIdPFQDN/SecureAuthIdPRealm/api/v1

## Definitions

**status:** The status of userId provided (found, not_found, invalid, etc.); will always be in response

**message:** Additional information regarding the status; will always be in response

**userId:** The user ID provided; will always be in response, whether successful or not

**properties:** The list of available user Profile Properties

- **firstName:** The user's First Name entered in the SecureAuth IdP Property / directory field
- **lastName:** The user's Last Name entered in the SecureAuth IdP Property / directory field
- **phone1-4:** The user's phone number(s) (Phone 1, Phone 2, Phone 3, Phone 4) entered in SecureAuth IdP Property / directory fields
- **email1-4:** The user's email address(es) (Email 1, Email 2, Email 3, Email 4) entered in SecureAuth IdP Property / directory fields
- **pinHash:** The user's PIN number saved in the SecureAuth IdP Property / directory field
- **auxId1-10:** The user's auxiliary ID content (Aux ID 1 - Aux ID 10) populated in the SecureAuth IdP Property / directory fields
- **ExtProperty1-##:** Information from the user's additional property fields populated in SecureAuth IdP Property / directory fields

**knowledgeBase:**

- **kbq1-6:** The user's knowledge-based questions and answers (1 - 6) written to the SecureAuth IdP data store
- **helpDeskKb:** The user's Help Desk knowledge-based question and answer written to the SecureAuth IdP data store

## GET endpoint

The **/users GET** endpoint retrieves a list of end user profile properties. SecureAuth IdP accesses and retrieves the user's profile from the username in the endpoint URL.

As a **GET** endpoint, there is no body, so JSON parameters are not required in the message body.

| GET Endpoint | Example |
|---|---|
| /users/{userId} | https://secureauth.company.com/secureauth2/api/v1/users/jdoe |

## Web Admin configuration notes

WebAdmin Configuration:

- The **isWritable** flag is configured in the WebAdmin **Data** tab.

Profile data edits:

- Profile data that responds to the GET request can be updated in the WebAdmin.
- To clear an attribute from the profile, include the schema name and an empty string.

## Successful GET Retrieval Information

GET retrieval is successful in the following situations:

- The retrieval includes a collection of usergroups that a user belongs to (LDAP only).
- Groups are retrieved with the FQDN (LDAP only).
- For the end user profile, attribute metadata is included, for example, display name, schema name, and whether or not the attribute is writable.
- If the end user does not have data in an attribute, the attribute will not appear in the profile list, even if it is mapped in the WebAdmin.
- The maximum number of properties that can be retrieved for each of the following attributes are:
    - "phone" = 4
    - "email" = 4
    - "auxId" = 10
    - "ExtProperty"# = 1 or more
    - knowledgeBase "kbq" = 6

## Definitions

**groups:**

- **CN=commonName,OU=organizationalUnitName,DC=domainComponent,DC=local**: End user Distinguished Name string
  e.g. **CN=SharePoint Developers,OU=jdoe,DC=dev,DC=local**

**accessHistories:**

- **userAgent:** End user client software identifier
- **ipAddress:** End user client IP address

- **timestamp:** Time the request was made
- **authState:** Request authorization status

## Response Examples

| Success Response | Failed Response / Error |
|---|---|
| <pre>{<br>  "userId": "jdoe",<br>  "properties": {<br>  "firstName": {<br>  "value": "John",<br>  "isWritable": "true"<br>  },<br>  "lastName": {<br>  "value": "Doe",<br>  "isWritable": "true"<br>  },<br>  "phone1": {<br>  "value": "123-456-7890",<br>  "isWritable": "true"<br>  },<br>  "phone2": {<br>  "value": "234-567-8910",<br>  "isWritable": "true"<br>  },<br>  "email1": {<br>  "value": "jdoe@dev.local",<br>  "isWritable": "true"<br>  },<br>  "email2": {<br>  "value": "jdoe@gmail.com",<br>  "isWritable": "true"<br>  },<br>  "pinHash": {<br>  "value": "1234",<br>  "isWritable": "true"<br>  },<br>  "auxId1": {<br>  "value": "123 Anywhere Drive",<br>  "isWritable": "true"<br>  },<br>  "auxId2": {<br>  "value": "Suite #100",<br>  "isWritable": "true"<br>  },<br>  "ExtProperty1": {<br>  "displayName": "New Property",<br>  "value": "John",<br>  "isWritable": "false"<br>  }<br>  },<br>  "knowledgeBase": {<br>  "kbq1": {<br>  "question": "what is your favorite color?",<br>  "answer": "red"<br>  },<br>  "kbq2": {<br>  "question": "What was your favorite childhood game?",<br>  "answer": "hide and seek"<br>  },<br>  "helpDeskKb": {<br>  "question": "What city were you born in?",<br>  "answer": "Atlanta"<br>  }<br>  },<br>  "groups": [<br>  "CN=SharePoint Developers,OU=jdoe,DC=dev,DC=local",<br>  "CN=SharePoint RnD,OU=jdoe,DC=admin,DC=local"<br>  ],<br>  "accessHistories": [<br>  {<br>  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4)"<br>  "ipAddress": "192.168.1.2",<br>  "timeStamp": "2016-04-12T22:14:19.928868Z",<br>  "authState": "Success"<br>  }</pre> | <pre>{<br>  "status": "not_found",<br>  "message": "User Id was not found"<br>}<br>HTTP Status 404</pre><br><pre>{<br>  "status": "invalid_group",<br>  "message": "User Id is not associated with a valid group."<br>}<br>HTTP Status 200</pre><br><pre>{<br>  "status": "disabled",<br>  "message": "Account is disabled."<br>}<br>HTTP Status 200</pre><br><pre>{<br>  "status": "lock_out",<br>  "message": "Account is locked out."<br>}<br>HTTP Status 200</pre><br><pre>{<br>  "status": "password_expired",<br>  "message": "Password is expired."<br>}<br>HTTP Status 200</pre><br>See Server Error information below |

## PUT / POST endpoints

The **/users PUT / POST** endpoints add, update, or delete end user profile properties. SecureAuth IdP updates the user's profile by using the username in the endpoint URL.

| PUT / POST Endpoint | Example |
|---|---|
| /users/{userId} | https://secureauth.company.com/secureauth2/api/v1/users/jdoe |

## Notes

- Extended properties cannot be updated.
- The userId is included in the URL, so is not a part of the request.
- The maximum number of properties that can be included in the message body for each of the following attributes are:
  - "phone" = 4
  - "email" = 4
  - auxId" = 10
  - "knowledgeBase" kbq" = 6

## Message body and response examples

| Message Body | Success Response | Failed Response | Error Response |
|---|---|---|---|

| | | | |
|---|---|---|---|
| <pre>{ ],<br>"properties": {<br>"firstName": "John",<br>"lastName": "Doe",<br>"phone1": "123-456-7890",<br>"phone2": "234-567-8910",<br>"email1": "jdoe@dev.local",<br>"email2": "jdoe@gmail.com",<br>"pinHash": "1234",<br>"auxId1": "123 Anywhere Drive"<br>"auxId2": "Suite #100"<br>},<br>"knowledgeBase": {<br>"kbq1": {<br>"question": "What is your favorite color?",<br>"answer": "red"<br>},<br>"kbq2": {<br>"question": "What was your favorite childhood game?",<br>"answer": "hide and seek"<br>},<br>"helpDeskKb": {<br>"question": "What city were you born in?",<br>"answer": "Alexandria"<br>}<br>}<br>}</pre> | <pre>{<br>"status": "found",<br>"message": ""<br>}</pre> | <pre>{<br>"status": "success"<br>"message": ""<br>}</pre> | <pre>{<br>"status": "failed"<br>"message": "Invalid username."<br>}</pre><br><pre>{<br>"status": "failed"<br>"message": "Invalid password."<br>}</pre><br><pre>{<br>"status": "failed"<br>"message": "Invalid email."<br>}</pre><br><pre>{<br>"status": "failed"<br>"message": "Provider error."<br>}</pre><br><pre>{<br>"status": "failed"<br>"message": "Duplicate username."<br>}</pre><br><pre>{<br>"status": "failed"<br>"message": "Duplicate email."<br>}</pre><br><pre>{<br>"status": "failed"<br>"message": "Unknown error."<br>}</pre> | <pre>{<br>"status": "error"<br>"message": "Not_Found"<br>}</pre> |

## POST endpoints

Use the POST endpoints to create an end-user profile, perform an administrative password reset for the end user, and enable end users to change their password when necessary.

### Create user

The **/users POST user** endpoint creates the new end user profile, so a username is not specified in the endpoint URL

| POST Endpoint | Example |
|---|---|
| /users/ | https://secureauth.company.com/secureauth2/api/v1/users/ |

## Notes

- The request is the same as the one used for update user profile, although you can specify a user ID and password.
- "Provider error" indicates a failure from the data provider.
- The user is created at the root of the connection string (LDAP only).
- To use a specific location for the user profile, the path to the correct OU must be specified in the **Connection String** on the **Data** tab.
- The maximum number of properties that can be included in the message body for each of the following attributes are:
  - "phone" = 4
  - "email" = 4
  - "auxId" = 10
  - knowledgeBase "kbq" = 6

## Message body and response examples

| Message Body | Success Response | Failed Response | Error Response |
|---|---|---|---|
| <pre>{<br>  "userId": "jdoe",<br>  "password": "93$q!SAT",<br>  "properties": {<br>  "firstName": "John",<br>  "lastName": "Doe",<br>  "phone1": "123-456-7890",<br>  "phone2": "234-567-8910",<br>  "email1": "jdoe@dev.local",<br>  "email2": "jdoe@gmail.com",<br>  "pinHash": "1234",<br>  "auxId1": "123 Anywhere Drive",<br>  "auxId2": "Suite #100"<br>  },<br>  "knowledgeBase": {<br>  "kbq1": {<br>  "question": "What is your favorite color?",<br>  "answer": "red"<br>  },<br>  "kbq2": {<br>  "question": "What was your favorite childhood game?",<br>  "answer": "hide and seek"<br>  },<br>  "helpDeskKb": {<br>  "question": "What city were you born in?",<br>  "answer": "Alexandria"<br>  }<br>  }<br>}</pre> | <pre>{<br>  "status": "success"<br>  "message": ""<br>}</pre> | <pre>{<br>  "status": "failed"<br>  "message": "Invalid username."<br>}</pre><br><pre>{<br>  "status": "failed"<br>  "message": "Invalid password."<br>}</pre><br><pre>{<br>  "status": "failed"<br>  "message": "Invalid email."<br>}</pre><br><pre>{<br>  "status": "failed"<br>  "message": "Provider error."<br>}</pre><br><pre>{<br>  "status": "failed"<br>  "message": "Duplicate username."<br>}</pre><br><pre>{<br>  "status": "failed"<br>  "message": "Duplicate email."<br>}</pre><br><pre>{<br>  "status": "failed"<br>  "message": "Unknown error."<br>}</pre> | <pre>{<br>  "status": "error"<br>  "message": "Not_Found"<br>}</pre> |

## Reset user password

The **/users POST resetpwd** endpoint performs an administrative password reset for the end user. SecureAuth IdP accesses the end user's profile, resets the user's password, and provides a new password by using the username in the endpoint URL.



The following POST endpoint calls SecureAuth API v2 in the example path. SecureAuth API v2 ignores the **userAccountControl** status, which enables the API to reset a user password.

- To use v2 with SecureAuth IdP v9.3.x, you must install Hotfix 9.3.0-15 or later.
- To use v2 with the SecureAuth® Identity Platform v19.07.xx, you must install Hotfix 19.07.01-8 or later.

If you require the SecureAuth API to block **userAccountControl** restrictions, continue to call v1. (**userAccountControl** ensures that certain account statuses are handled appropriately in transactions between LDAP providers and SecureAuth IdP/Identity Platform.)

| POST Endpoint | Example |
|---|---|
| /users/{userId}/resetpwd | https://secureauth.company.com/secureauth2/api/v2/users/jdoe/resetpwd |

# Notes

- The current password does not need to be provided by the administrator.
- A failed response references the text set up in the **contextuser_changepwd 1-4** fields in the Web Admin.
  - The administrator can edit the text for these fields in the Verbiage Editor in the Web Admin.
  - To access the Verbiage Editor, open the **Overview** tab and click the **Content and Localization** link.

# Message body and responses

| Message Body | Success Response | Failed Response |
|---|---|---|
| <pre>{<br>"password": "M@g1cHappens"<br>}</pre> | <pre>{<br>"status": "success"<br>"message": "Password was reset"<br>}</pre> | <pre>{<br>"status": "failed"<br>"message": ""<br>}</pre> |

### User self-service change password

The **/users POST changepwd** endpoint performs a password reset for the end user. SecureAuth IdP accesses the end user's profile and lets the end user change that password by using the username in the endpoint URL.

| POST Endpoint | Example |
|---|---|
| /users/{userId}/changepwd | https://secureauth.company.com/secureauth2/api/v1/users/jdoe/changepwd |

## Notes

- The end user must provide the existing password to change the password.
- A failed response references the text set up in the **contextuser_changepwd1-4** fields in the Web Admin.
  - The administrator can edit the text for these fields in the **Verbiage Editor** in the Web Admin.
  - To access the Verbiage Editor, open the **Overview** tab and click the **Content and Localization** link.

## Message body and response examples

| Messsage Body | Success Response | Failed Response |
|---|---|---|
| <pre>{<br>"currentPassword": "M@g1cHappens",<br>"newPassword": "D3fault321"<br>}</pre> | <pre>{<br>"status": "success"<br>"message": "Password was changed"<br>}</pre> | <pre>{<br>"status": "failed"<br>"message": ""<br>}</pre> |

# Configure group association endpoints

Admins can use **POST** messages to associate users with groups and vice-versa.

Types of associations to the **/users** or **/groups** endpoint include the following:

- Single user to single group
- Single user to multiple groups
- Single group to single user
- Single group to multiple users

## Single user to single group

This operation associates a single user in the data store with a single group in the data store.

No message body is required because all parameters for this request are present in the call URL.

| POST Endpoint | Example |
|---|---|
| /users/{userId}/groups/{groupID to associate} | https://secureauth.company.com/secureauth2/api/v1/users/jdoe/groups/admins |

**Result**: userID "jdoe" is associated with the groupID "admins"

## Response examples

| | |
|---|---|

| Success Response | Failure Response |
|---|---|
| <pre>{<br>    "status": "success"<br>    "message": ""<br>}</pre> | <pre>{<br>    "status": "failure"<br>    "message": "Failed to add user to group."<br>}</pre> |
| | <pre>{<br>    "status": "failure"<br>    "message": "Group actions are not supported with the current configuration."<br>}</pre> |

## Single group to multiple users

This operation associates a list of multiple users with a specified group.

For multiple users, supply the list of users in the POST message body, not the URL.

If any of the userIDs fail to POST, a failure response is generated that lists each userID that failed. userIDs not listed in the failure response successfully POSTed.

| POST Endpoint | Example |
|---|---|
| /users/{userId}/groups/{groupId}/users | https://secureauth.company.com/secureauth2/api/v1/groups/Sharepoint%20Visitors/users |

**Result**: group "Sharepoint Visitors" is associated with the list of users specified in the message body

## Message body and response examples

| Message Body | Success Response | Failure Response |
|---|---|---|
| <pre>{<br>    "userIds" : [<br>        "jdoe",<br>        "jsmith",<br>        "kmartin",<br>        "pjohnson",<br>    ]<br>}</pre> | <pre>{<br>    "status": "success"<br>    "message": ""<br>}</pre> | <pre>{<br>  "failures": {<br>    "Sharepoint Visitors": [<br>      "kmartin",<br>      "pjohnson"<br>    ]<br>  },<br>  "status": "failed",<br>  "message": "There were 2 association errors."<br>}</pre> |

## Single group to single user

This operation associates a group in the data store with a single user in the data store; the operation is functionally equivalent to the Single User to Single Group operation.

No message body is required because all parameters for this request are present in the call URL.

| POST Endpoint | Example |
|---|---|
| /groups/{groupID}/users/{userID to associate} | https://secureauth.company.com/secureauth2/api/v1/groups/admins/users/jdoe |

**Result**: groupID "admins" is associated with userID "jdoe"

## Response examples

| Success Response | Failure Response |
|---|---|
| <pre>{<br>    "status": "success"<br>    "message": ""<br>}</pre> | <pre>{<br>    "status": "failure"<br>    "message": "Failed to add user to group."<br>}</pre> |
| | |

| | |
|---|---|
| | <pre>{<br>    "status": "failure"<br>    "message": "Group actions are not supported with the current configuration."<br>}</pre> |

## Single user to multiple groups

This operation associates a single user with multiple groups at the same time.

For multiple groups, supply the list of groups in the POST message body, not the URL.

If any of the groupIDs fail to POST, a failure response is generated that lists each groupID that failed. groupIDs not listed in the failure response successfully POSTed.

| POST Endpoint | Example |
|---|---|
| /users/{userId}/groups | https://secureauth.company.com/secureauth2/api/v1/users/jdoe/groups |

**Result**: user "jdoe" is associated with a list of groups specified in the message body

## Message body and response examples

| Message Body | Success Response | Failure Response |
|---|---|---|
| <pre>{<br>   "groupNames" : [<br>       "SharePoint Visitors",<br>       "SharePoint Developers"<br>   ]<br>}</pre> | <pre>{<br>    "status": "success"<br>    "message": ""<br>}</pre> | <pre>{<br>  "failures": {<br>    "jdoe": [<br>      "SharePoint Visitors",<br>      "SharePoint Developers"<br>    ]<br>  },<br>  "status": "failed",<br>  "message": "There were 2 association errors."<br>}</pre> |

# Server error

A server error returns the following response:

```
{
  "status": "server_error",
  "message": "<Exception message describing the issue.>",
}
HTTP Status 500
```