AccessMatrix™

# SecurLogin REST API Reference

i-Sprint
Trust without Boundaries

**Document Version: 7.1**

**Date: 31 March 2022**



**TRADEMARK INFORMATION and DISCLAIMER**

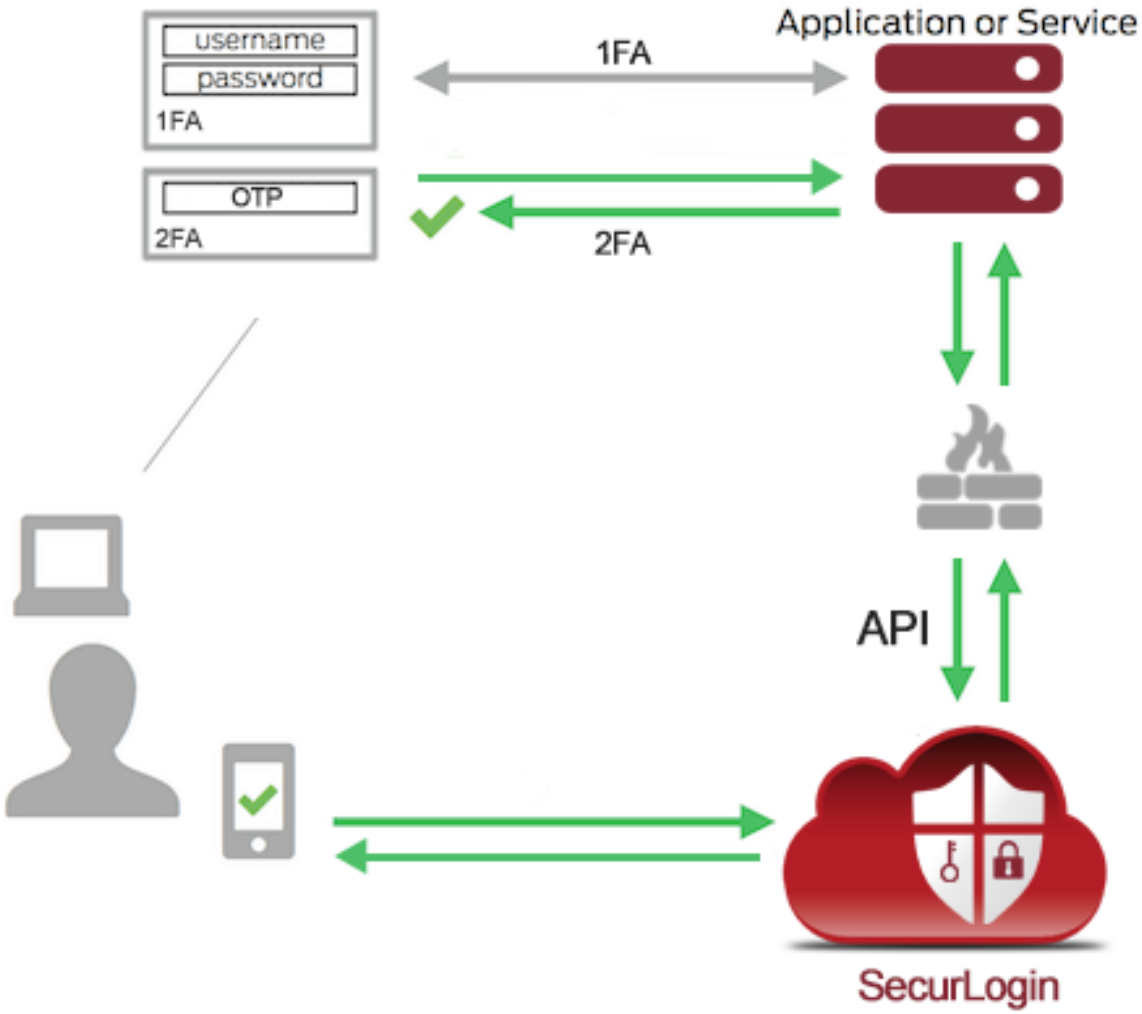
# Table of Contents



# API Introduction

The API is a low-level, RESTful API for tenant applications to access SecurLogin to do 2fa authentication or to make the administrative management operations.



# Prerequisite

Before you start using any APIs, you need:

- Sign up for a SecurLogin tenant account.
- Log in to the SecurLogin Administration portal and navigate to "Applications".
- Click "Protect an Application" and select "API" as the Application Type. Click "Save" then you will get your Application Key and Secure key.

The Application Key and Secure key will be used as API authentications.

## API Authentication

RESTful API Authentication is the process of proving your identity to the system. Identity is an important factor in SecurLogin access control decisions. Requests are allowed or denied in part based on the identity of the requester.
SecurLogin API is RESTful API that uses HTTP requests to GET, PUT, POST and DELETE data. The SecurLogin REST API is based on HTTP Basic authentication (BA) implementation which leverages standard fields in the HTTP Header, obviating the need for handshakes.A keyed-HMAC (Hash Message Authentication Code) is used for authentication.

### Signature Based Authentication

To authenticate a request, you first concatenate selected elements of the request to form a string. You then use your SecurLogin secure key to calculate the HMAC of that string. Informally, we call this process "signing the request," and we call the output of the HMAC algorithm the signature, because it simulates the security properties of a real signature. Finally, you add this signature as a parameter of the request by using the syntax described in this doc.

### Request Header Format

SecurLogin API authenticate data by header that is following standard HTTP. Header format is as below.

```
Authorization = "Basic" + " " + Base64.encode(applicationKey+ ":" + Signature);
Signature = HMAC-SHA256(secureKey, UTF8-Encoding-Of(StringToSign)).toupper();
StringToSign = requestTime + "\n" + method + "\n" + path + "\n" + params;
params="username=abc&userstore_id=xxx";
```

For above format

1. applicationKey and secureKey are generated and assigned to apps by SecurLogin automatically.
2. Unicode code for “\n” is 0x0A which is known as line break.
3. HMAC-SHA256 is a standard HMAC authentication algorithm based on message. It requires 2 kinds byte-strings input, i.e. 1) a secureKey for signature; 2) a message body that will be signed. A message summary will be generated as the output of such algorithm.
4. To construct the StringToSign, first build an ASCII string from your request, using the following items:
5. The request time, formatted as RFC 2822. This must be the same string as the "Date" header.

| Parameter | Description | Example |
|---|---|---|
| requestTime | The request time, formatted as RFC 2822. This must be the same string as the "Date" header. It must be English and following the expressions like the example. | Tue, 01 Jun 2016 10:19:31 -0000 |
| method | The HTTP method (uppercase) | POST |
| path | The specific API method's path | auth1/v1_2/preauth |
| params | The URL-encoded list of “key=value” pairs, lexicographically sorted by key. These come from the request parameters (the URL query string for GET request or the request body for POST requests). If the request does not have any parameters one must still include a blank line in the string that is signed. | username=abc&userstore_id=xxx |

### Authentication Examples

The examples in this section use the (non-working) credentials in the following table.

| Parameter | Value |
|---|---|
| applicationKey | 3pL7zOgLTuirfnH5DDs |
| secureKey | nEtETyIRlgQXFbYJ6wVvPUnUsyHFB5Ac2v3rhLyJNghlzmDdcKwxvyE9Gev0duC |

In the example StringToSigns, formatting is significant, and \n means the Unicode code point U+000A, commonly called newline.

**Example Object POST**
This is an example of the /preauth API.

| **Parameter** |
| --- |
| POST /tenant/v2_0/preauth  HTTP/1.1<br>Content-Type: application/x-www-form-urlencoded<br>Content-Length: 142<br>Encoding: UTF-8<br>Host: api.securlogin.com<br>Date: Tue, 01 Jun 2016 10:19:31 -0000<br><br>Authorization:Basic<br>M3BMN3pPZ0xUdWlyZm5INUREczozQzI4QzkwMjVCNTg4M0M3MzhEMDhGNEFGOEMwQUI4RTdDMEU2RDQ4NDkzQjVBMzdCQkVENjJGNjlDQTY2<br><br>username=abc&userstore_id=xxx |
| **StringToSign** |
| Tue, 01 Jun 2016 10:19:31 -0000\n<br>POST\n<br>tenant/v2_0/preauth\n<br>username=abc&userstore_id=xxx |
| **Signature** |
| 8E9F4F5D745EFCDE37E11CDA26F82B2B4BEFA92B831DAF4BD5FF144AEC995054 |
| **Authorization** |
| Basic<br>M3BMN3pPZ0xUdWlyZm5INUREczo4RTlGNEY1RDc0NUVGQ0RFMzdFMTFDREEyNkY4MkIyQjRCRUZBOTJCODMxREFGNEJENUZGMTQ0QUVDO |

## Example Code to Signature

The example code in different language is in the following table.

**Java**

```
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Locale;
import java.util.TimeZone;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import net.sf.json.JSONObject;
public class SignUtil {
 private static CloseableHttpClient httpclient = HttpClients.createDefault();
 private static String url = "https://api.securlogin.com";
 private static String applicationKey = "3pL7zOgLTuirfnH5DDs";
 private static String secureKey = "nYp97le3KRf6cElsLnC8zM340sZ9zv4AEQe0VT5BhD15p2LLjX6UaVQ9MYIxwKnH";

 public static void main(String[] args) throws UnsupportedEncodingException {
  String path = "tenant/v2_0/preauth";
  String ping_url = "/SecurLoginAPI/webservice/"+path;
  JSONObject obj = new JSONObject();
  obj.put("username", "demoUser");
  obj.put("userstore_id", "xxx");
  System.out.println(url+ping_url);
  String response = executePostRequestForSecurLogin(url+ping_url,path,obj.toString());
  System.out.println("response = " + response);
 }
 public static String executePostRequestForSecurLogin(String url, String path, String postBody) throws UnsupportedEncodingException {

       String paras = JSONObject.fromObject(postBody).toString();

  Calendar cal = Calendar.getInstance();

  SimpleDateFormat greenwichDate = new SimpleDateFormat("EEE, d MMM yyyy HH:mm:ss 'GMT'", Locale.US);
```

```
 greenwichDate.setTimeZone(TimeZone.getTimeZone("GMT"));
 String requestTime = greenwichDate.format(cal.getTime());
 String sign = "";
 String sign1 = requestTime + "\n" + "POST" + "\n" + path + "\n" + paras;
 try {
  sign = (sign(secureKey.getBytes("UTF-8"), sign1.getBytes()   )   ).toUpperCase();
  System.out.println(sign);
 } catch (InvalidKeyException e1) {
  e1.printStackTrace();
 } catch (NoSuchAlgorithmException e1) {
  e1.printStackTrace();
 }

 String plainClientCredentials2 = applicationKey + ":" + sign;
 String base64ClientCredentials2 = new String(Base64.encodeBase64String(plainClientCredentials2.getBytes()));
 HttpPost httpPost = new HttpPost(url);
 httpPost.setHeader("Authorization", "Basic " + base64ClientCredentials2);
 httpPost.setHeader("Charset", "UTF-8");
 httpPost.setHeader("Content-Type", "application/x-www-form-urlencoded");
 postBody = "username=demoUser";
 StringEntity entity = new StringEntity(postBody, "UTF-8");

 httpPost.setHeader("Date", requestTime);

 httpPost.addHeader("Accept-Language",
   Locale.getDefault().getLanguage().toLowerCase() + "-" + Locale.getDefault().getCountry().toLowerCase());
 httpPost.addHeader("Local-Region",
   Locale.getDefault().getLanguage().toLowerCase() + "-" + Locale.getDefault().getCountry().toLowerCase());
 httpPost.setEntity(entity);
 HttpResponse response = null;
 String response1 = null;
 try {
  response = httpclient.execute(httpPost);
  response1 = EntityUtils.toString(response.getEntity(), "UTF-8");
 } catch (IOException e) {
  e.getMessage();
 }
 return response1;
}


 public static String sign(byte [] data, byte [] key) throws NoSuchAlgorithmException, InvalidKeyException {

  Mac sha256_HMAC = Mac.getInstance("HmacSHA256");
  SecretKeySpec secret_key = new SecretKeySpec(key, "HmacSHA256");
  sha256_HMAC.init(secret_key);

  return byte2hex(sha256_HMAC.doFinal(data));
 }

 public static String byte2hex(byte[] b) {
       String hs = "";
       String stmp = "";
       for (int n = 0; n < b.length; n++) {
          stmp = (java.lang.Integer.toHexString(b[n] & 0XFF));
          if (stmp.length() == 1) {
             hs = hs + "0" + stmp;
          } else {
             hs = hs + stmp;
          }
       }
       return hs.toUpperCase();
    }
}
```

**nodejs**

```
var _        = require('lodash');
var crypto   = require('crypto');
var axios    = require('axios');
const util = require('util')

const date = 'Tue, 01 Jun 2016 10:19:31 -0000';

const applicationKey = "3pL7zOgLTuirfnH5DDs";

const secureKey = "nEtETyIRlgQXFbYJ6wVvPUnUsyHFB5Ac2v3rhLyJNghIzmDdcKwxvyE9Gev0duCg";

const StringToSign = _.template("<%= requestTime %>\n<%= method %>\n<%= path %>\n<%= params %>");

const hostName = "https://api.securlogin.com/SecurLoginAPI/webservice";
```

```
const preAuth = async (userName) => {

    const jsonParams = {
        username: userName
    }

    const params = new URLSearchParams((jsonParams)).toString();

    let StringToSignEncoded = StringToSign({
        method: "POST",
        path: "tenant/v2_0/preauth",
        requestTime: date,
        params: params
    });

    const Authorization = getAuthorizationSignedString(StringToSignEncoded);

    const HeaderConfig = {
        headers: {
            Authorization,
            date
        }
    }

    await axios.post(`${hostName}/tenant/v2_0/preauth`, params, HeaderConfig)
        .then((response) => {
            console.log((util.inspect(response, {showHidden: false, depth: null, colors: true})));
        });
}

const getAuthorizationSignedString = (StringToSignEncoded) => {
    const Signature = crypto.createHmac('sha256', secureKey).update(StringToSignEncoded).digest('hex').toUpperCase();

    const buff64 = Buffer.from(applicationKey + ":"+Signature).toString('base64');

    const Authorization = "Basic" + " " + buff64;

    return Authorization;
}

module.exports = {
preAuth
};

)))
```

# Authentication API

## /ping

Access Type: GET
Access Path: /tenant/v2_0/ping
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/ping

The /ping is used for check that SecurLogin service is up and running before calling to other APIs. Note that this one does not need to be signed with the Authorization header.

**Request**
None

**Response**

| Parameter | Value |
|---|---|
| time | Current server time. Formatted as a UNIX timestamp. Integer. |

JSON example:

```
{
 "status": "OK",
 "response": {
   "time": 1487927603
  }
}
```

## /check

Access Type: GET
Access Path: /tenant/v2_0/check
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/check

The /check API can be called to verify that the applicationKey and secureKey are valid, and that the signature is being generated correctly.

**Request**
None

**Response**

| Parameter | Value |
|---|---|
| time | Current server time. Formatted as a UNIX timestamp. Integer. |

JSON example:

```
{
 "status": "OK",
 "response": {
   "time": 1487927603
  }
}
```

## /preauth

Access Type: Post
Access Path: tenant/v2_0/preauth
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/preauth
Content-Type: application/x-www-form-urlencoded

The /preauth API checks user's policy and determines whether the user is authorized to log in, and then returns the user's available authentication factors, including push, token, email, SMS, phone.

**Request**

| Parameter | Required | Description |
|---|---|---|
| user_uuid | See Description | Global unique identifier for the user. Generated by SecurLogin. Either User's UUID or the username is required to be provided. |

| | | |
|---|---|---|
| username | See Description | The user name to identify the user. It's unique value per user store. Either User's UUID or the username is required to be provided. |
| userstore_id | Optional | User store ID, normally the userstore is mapped with application, in case the mapping relationship is missing, you can pass in the userstore ID. |
| user_ip | Optional | User IP. |

**Response**

| Parameter | Value |
|---|---|
| result | One of the following strings:<br>**Value** / **Description**<br>auth: The user is allowed to proceed the SecurLogin authentication. You can call the /auth API to perform authentication<br>bypass: The user is configured to bypass SecurLogin authentication. You can immediately grant access without 2fa.<br>deny: The user is not permitted to authenticate at this time. You can immediately deny access.<br>enroll: The user is not enrolled to SecurLogin and needs to enroll. You should deny access and ask user to enroll. |
| message | This string is intended for display to the user. |
| 2fa_methods | This field will only be applicable if result is "auth".<br>This is a list of the user's 2fa methods, and each method is a series of key/value pairs.<br>**Key** / **Value**<br>method: List of strings, each a factor that can be used with the device.<br>**Method** / **Description**<br>push: SecurLogin Push.<br>sms: SMS OTP.<br>phone: Phone call OTP.<br>token: OTP generated in SecurLogin Mobile app.<br>email: Email OTP.<br>name: The display name of the 2fa method.<br>device: Phone value of the device id or email or phone number. (optional for token method).<br>enroll_time: The registration time of email and phone number, or the download token time, and the format is yyyy-MM-dd hh:mi:ss. |

JSON example:

If the result is "auth":

```
{
 "status": "OK",
 "response": {
   "result": "auth",
   "message": "Please choose a 2fa method",
   "2fa_methods": [
      {
       "method": "phone",
       "enroll_time": "2017-01-03 12:02:34",
       "device": "008618666906094",
       "name": "Wayn's phone1"
      },
      {
       "method": "token",
       "enroll_time": "2017-01-04 12:02:34",
       "device": "SFDE3FE3451332FDFV",
```

```
       "name": "i-Sprint Token"
      },
{
       "method": "email",
       "enroll_time": "2017-01-05 12:02:34",
       "device": "wayne.wang@i-sprint.com",
       "name": "i-Sprint Email"
      }
    ]
  }
}
```


If the result is "enroll":


```
{
 "status": "OK",
 "response": {
"result": "enroll",
      "message": "Please enroll an authentication method to proceed"
  }
}
```


## /auth_0

Access Type: Post
Access Path: /tenant/v2_0/auth_0
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/auth_0
Content-Type: application/x-www-form-urlencoded

The /auth_0 API performs second-factor authentication for a user by sending a push notification to the user's smartphone app, sending a phone call/SMS/Email OTP.

**Request**

| Parameter | Required | Description |
|---|---|---|
| method | Required | The following authentication methods are supported.<br><br>**Capability** / **Description**<br>2 / SecurLogin Push(push).<br>3 / SMS OTP(sms).<br>4 / Phone call OTP(phone).<br>5 / Email OTP(email). |
| user_uuid | See Description | Global unique identifier for the user. Generated by SecurLogin. Either User's UUID or the username is required to be provided. |
| username | See Description | The user name to identify the user. It's unique value per user store. Either User's UUID or the username is required to be provided. |
| userstore_id | Optional | User store ID, normally the userstore is mapped with application, in case the mapping relationship is missing, you can pass in the userstore ID. |
| user_ip | Optional | User IP. |
| device | Optional | Refer to the "device" field in /preauth API. If the value is null, by default, the OTP will be sent to the last registered email or phone number according to the authentication method, or the push message will be sent to the last download token. |
| pushinfo | Optional | Only applicable for push authentication method.<br>A set of URL-encoded key/value pairs with additional contextual information associated with this authentication attempt. The SecurLogin Mobile app will display this information to the user.<br><br>For example: from=login %20portal&domain=example.com<br><br>The URL-encoded string's total length must be less than 20,000 bytes. |

**Response**

| Parameter | Description |
|---|---|
| txid | A transaction ID to be used to call /authn_1 API or to query the authentication status using the /auth_status API. |
| expiry | Time at which this authentication request will be expired. Formatted as a UNIX timestamp. Integer. |

JSON EXAMPLE:

```
{
 "status": "OK",
 "response": {
   "txid": "SDFCERVER213242VF23424SDWEF34",
   "expiry": 1487927603
  }
}
```

# /auth_1

Access Type: Post
Access Path: /tenant/v2_0/auth_1
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/auth_1
Content-Type: application/x-www-form-urlencoded

The /auth_1 API performs second-factor authentication for verifying PhoneCall/SMS/Email/Token OTP.

**Request**

| Parameter | Required | Description |
|---|---|---|
| method | Required | The following authentication methods are supported.<br>Capability: 1 — OTP generated in SecurLogin Mobile app.(token)<br>3 — SMS OTP.(sms)<br>4 — Phone call OTP.(phone)<br>5 — Email OTP.(email) |
| otp | Required | OTP entered by user. |
| txid | Required(when the method is “phone”, “sms”, or “email”) | |
| device | Optional | Refer to the “device” field in /preauth API. Verify the device value(tokenSN)and OTP when method is token and device value is provided. Either device(tokenSN) or the username is required to be provided. |
| username | Optional | The user name to identify the user. It’s unique value per user store. Either device(tokenSN) or the username is required to be provided when method is token. |
| userstore_id | Optional | User store ID, normally the userstore is mapped with application, in case the mapping relationship is missing, you can pass in the userstore ID. If there is no value provided, default is internal users. |

**Response**

| Parameter | Description |
|---|---|
| result | Either "allow" or "deny". If "allow" is returned, your application should grant access to the user. If "deny", it should not. |
| message | The detailed message of the authentication attempt. If the authentication attempt was denied, it may identify a reason. This string is intended for display to the user. |

JSON EXAMPLE:

```
{
 "status": "OK",
```

```
 "response": {
   "result": "allow",
   "message": "SecurLogin authentication successful."
  }
}
```


## /auth_status

Access Type: GET
Access Path: /tenant/v2_0/auth_status/{txid}
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/auth_status/{txid}

The /auth_status API is used to poll for the next status update from the authentication process for a given transaction.

**Request**

| Parameter | Required | Description |
|---|---|---|
| txid | Required | The transaction ID of the authentication attempt, returned by the /auth API. |

**Response**

| Parameter | Description |
|---|---|
| result | One of the following values:<br>**Value** / **Description**<br>1 / Authentication succeeded. (allow)<br>2 / Authentication fail. (fail)<br>3 / Authentication denied because invalid. (invalid)<br>4 / Authentication denied because mistake. (mistake)<br>5 / Authentication is still in-progress. (waiting)<br>6 / Authentication time out. (timeout) |
| status | The detailed status of the result is “waiting”.<br>**Value** / **Description**<br>sent / SMS/Email OTP has been sent to user.<br>called / Phone call sent to user.<br>pushed / A SecurLogin push authentication request has been sent to the device.<br>failed / An error occurred while sending the notification to the user's device. The user should retrieve the request manually in the SecurLogin Mobile app.<br>timeout / Authentication timed out. |
| message | The detailed message of the authentication attempt. If the authentication attempt was denied, it may identify a reason. |

JSON EXAMPLE:


```
{
 "status": "OK",
 "response": {
   "result": "waiting",
   "status": "pushed",
   "message": "Pushed a login request to your phone."
  }
}
```


## /pre_enroll

Access Type: POST
Access Path: /tenant/v2_0/pre_enroll
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/pre_enroll
Content-Type: application/x-www-form-urlencoded

The /pre_enroll is to check if the user account is already enrolled in SecurLogin, and if it’s not enrolled before, you can call the /enroll to create the user ID in SecurLogin.

**Request**

| Parameter | Required | Description |
|---|---|---|
| username | Required | Unique identifier for the user. |
| userstore_id | Optional | User store ID, normally the userstore is mapped with application, in case the mapping relationship is missing, you can pass in the userstore ID. |

**Response**

| Parameter | Description |
|---|---|
| enrolled | true/false |
| user_uuid | This field will only be applicable if enrolled is "true".<br>If user ID has been enrolled before, return the user's uuid. |
| 2fa_methods | This field will only be applicable if enrolled is "true".<br>This is a list of the enrolled 2fa methods, and each method is a series of key/value pairs.<br>**Key** / **Value**<br>method: List of strings, each a factor that can be used with the device.<br>**method** / **Description**<br>phone: Phone call OTP.<br>token: OTP generated in SecurLogin Mobile app.<br>email: Email OTP.<br>name: The display name of the 2fa method.<br>device: Phone value of the device id or email or phone number. (optional for token method). |
| message | |

JSON EXAMPLE:

```
{
"status": "OK",
 "response": {
"enrolled": true,
"user_uuid": ES2OWMOF34R092VCOOCIES
"2fa_methods": [
      {
       "method": "phone",
       "device": "008618666906094",
       "name": "Wayne's phone1"
      },
      {
       "method": "token",
       "device": "SFDE3FE3451332FDFV",
       "name": "i-Sprint Token"
      },
 {
       "method": "email",
       "device": "wayne.wang@i-sprint.com",
       "name": "i-Sprint Email"
      }
]
"message": "The user ID has been enrolled before."
}
}
```

# /enroll_0

Access Type: POST
Access Path: /tenant/v2_0/enroll_0
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/enroll_0
Content-Type: application/x-www-form-urlencoded

The /enroll_0 is used to allow the application to choose which enrollment method is preferred.
User is either to use their phone number or email to receive the enrollment OTP, or scan the QR Code using the SecruLogin App to attach a 2FA method to the user account. For OTP enrolment you will need to call the /enroll_1 to submit the enrollment OTP.

For QRCode enrollment, subsequently, you can call the /enroll_status to check whether the enrollment is completed.

**Request**

| Parameter | Required | Description |
|---|---|---|
| username | Required | Unique identifier for the user. |
| userstore_id | Optional | User store ID, normally the userstore is mapped with application, in case the mapping relationship is missing, you can pass in the userstore ID. |
| group_id | Optional | User's group ID. |
| user_ip | Optional | User's IP. |
| method | Required | The following enrollment methods are supported.<br>Method: 1 – Email enrollment link.(email); 2 – SMS OTP.(sms); 3 – Phone call OTP.(phone); 4 – QR Code scan.(qrcode) |
| device | Optional | Email or phone number. Is required when the method is "phone", "sms" or "email". |

**Response**

| Parameter | Description |
|---|---|
| txid | A transaction ID to be used to query the enrollment status using the /enroll_status API. |
| qr_code | This field will only be applicable if method is "qrcode".<br>A QR Code used for SecurLogin mobile app to scan to finish the enrollment. |
| expiry | Time at which this enrollment will be expired. Formatted as a UNIX timestamp. Integer. |

JSON EXAMPLE:

```
{"status": "OK",
 "response": {
"txid": "SDFCERVER213242VF23424SDWEF34"
"qr_code": "https://www.securlogin.com/client/enroll_1?txid=SDFCERVER213242VF23424SDWEF34",
   "expiry": 1487927603
}
}
```

Failure case:

```
{
"status": "FAIL",
 "code": "60001",
 "message": "The phone or email have been used."
}
```

# /enroll_1

Access Type: POST
Access Path: /tenant/v2_0/enroll_1
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/enroll_1
Content-Type: application/x-www-form-urlencoded

The /enroll_1 is used to submit the OTP received via email or sms or phone call, after OTP is verified, a new user account will be enrolled with SecurLogin two-factor authentication attached.
Subsequently, you can call the /enroll_status to check whether the enrollment is completed.

**Request**

| Parameter | Required | Description |
|---|---|---|
| otp | Required | OTP input by user |
| txid | Required | |
| device | Optional | phone number. Is required when the method is "qrcode". |

**Response**

| Parameter | Description |
|---|---|
| result | One of the following strings:<br>**Value** / **Description**<br>completed – The user successfully enrolled.<br>invalid – The txid is expired or invalid. |
| message | |

JSON EXAMPLE:

```
{
"status": "OK"
"response": {
"result": "completed",
 "message": "The user successfully enrolled."
  }
}
```

# /enroll_status

Access Type: POST
Access Path: /tenant/v2_0/enroll_status/{txid}
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/enroll_status/{txid}
Content-Type: application/x-www-form-urlencoded

The /enroll_status is used to check whether a user has completed download the token.

**Request**

| Parameter | Required | Description |
|---|---|---|
| txid | Required | The transaction ID of the enrollment, returned by the /enroll_0 API. |

**Response**

| Parameter | Description |
|---|---|
| result | One of the following strings:<br>**Value** / **Description**<br>completed – The user successfully enrolled.<br>invalid – The txid is expired or not valid.<br>in_progress – The enrollment is still valid and has not yet completed. |
| status | The detailed status of the result is “waiting”.<br>**Value** / **Meaning**<br>sent – SMS OTP/Email has been sent to user.<br>called – Phone call sent to user.<br>scanned – The QRCode has been scanned by user but not yet approved to enroll.<br>timeout – Enrollment timed out. |
| message | The detailed message of the enrollment. If the enrollment was denied, it may identify a reason. |

JSON EXAMPLE:

```
{
 "status": "OK",
 "response": {
   "result": "in_progress",
   "status": "sent",
   "message": "OTP has been sent to your phone."
  }
}
```

# Administration API

The Admininitration API provides programmatic access to the administrative functionalities of SecurLogin cloud platform.

The Admin API API has methods for creating, retrieving, updating, and deleting the core objects in SecurLogin system such as users, devices, applications, policies and etc.

## /application_info

Access Type: GET
Access Path: /tenant/v2_0/application_info
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/application_info
Content-Type: application/x-www-form-urlencoded

/application_info API is used to obtain the special configuration for apps, e.g.AD configurations for Radius protocol apps.

**Request**

| Parameter | Required | Description |
|---|---|---|
| applicationKey | Required | applicationKey is generated by SecurLogin during app integration, and it should be provided to Radius proxy. |

**Response**

| Parameter | Description |
|---|---|
| applicationID | |
| applicationName | Application name. |
| use1Fa | The value can be enable or disable(default: disable). Static password authentication is no need if disable is returned. Otherwise, static password authentication is required, and AD configuration will be returned. |
| sendOtpType | 1-token;2-push;3-sms;4-call;5-email |
| primarySharedSecret | Return the encrypted Radius key. |
| radiusClientPort | Radius port |
| adUrl | ldap://192.168.1.1:389 LDAP address |
| useSsl | Default: true; SSL is applied or not. |
| loginID | Return the encrypted LDAP Administrator account |
| password | Return the encrypted LDAP Administrator password |
| adUserIdAttribute | AD account Attribute |
| adSearchContext | The search range of AD user |
| secondaryUrl | Secondary LDAP Address |
| secondaryUseSsl | |
| secondaryLoginID | Return the encrypted secondary LDAP Administrator account |
| secondaryPwd | Return the encrypted secondary LDAP Administrator password |
| securityGroupDN | |

JSON EXAMPLE:
Success:

```
{
 "status": "OK",
 "response": {
"applicationID": "123456",
"applicationName": "securlogin",
"primarySharedSecret": "Primary Radius shared secret" ,
       "radiusClientPort": "9999" ,
"adUrl": "ldap://192.168.1.1:389",
"useSsl": "true",
"loginID": "user1",
"password": "pwd123",
"secondaryUrl": "Optional",
"secondaryUseSsl": "Optional",
"secondaryLoginID": "Optional",
"secondaryPwd": "Optional",
"securityGroupDN": "Optional",
"adUserIdAttribute": "sAMAccountName",
"adSearchContext": "DC=AnXunBen,DC=COM",
"use1Fa": "enable",
```

```
"sendOtpType": "3"
  }
}
```

Failure:

```
{
 "status": "FAIL",
 "code": "2008",
 "message": "Invalid applicationKey. Unable to obtain securetKey by this applicationKey."
}
```

# /user_store_info

Access Type: GET
Access Path: tenant/v2_0/user_store_info
Access URL: https://api.securlogin.com/SecurLoginAPI/webservice/tenant/v2_0/user_store_info
Content-Type: application/x-www-form-urlencoded

/user_store_info API is used to obtain the synced database info.

**Request**

| Parameter | Required | Description |
|---|---|---|
| userstore_id | Required | It is offered to proxy in advance by other method such as email. |

**Response**

| Parameter | Required | Description |
|---|---|---|
| userStoreType | Required | Supports JDBC and AD currently. |
| dbType | Optional | userStoreType=For JDBC (Required), MySQL or Oracle will be returned. For userStoreType=AD, nothing will be returned. |
| url | Optional | URL example: ldap://192.168.1.1:389 189:3306/securlogin_clouddb |
| loginID | Optional | User ID used for login client database or AD (will be returned after encryption) |
| password | Optional | User password used for login client database or AD (will be returned after encryption) |
| userTableName | Optional | User table name of client database. It will be returned if userStoreType=JDBC (Required), and will not be returned if userStoreType=AD. |
| userUID | Optional | Primary key field name of client database user table.It will be returned if userStoreType=JDBC (Required), and will not be returned if userStoreType=AD. |
| userAttributesMapping | Optional | The fields mapping (HashMap<String, String>) between SecurLogin user table and client user table (or AD system). |
| groupTableName | Optional | User group table name of client database user table. It will be returned if userStoreType=JDBC (Required), and will not be returned if userStoreType=AD. |
| groupUID | Optional | Primary key field name of client database user group table.It will be returned if userStoreType=JDBC (Required), and will not be returned if userStoreType=AD. |
| groupAttributesMapping | Optional | The fields mapping between SecurLogin user group table and client user group table. It will be returned if userStoreType=JDBC (Required), and will not be returned if userStoreType=AD. |
| reconSchedule | Optional | Regular Expression that obtain the time of call of getUserStoreConfig. Example: 0 0/1 * 1/1 * ? * |
| useSSL | Optional | SSL protocol is used or not.It will not be returned if userStoreType=JDBC (Required), and will be returned if userStoreType=AD. |
| ntDomain | Optional | The domain name of client AD system. It will not be returned if userStoreType=JDBC |

| | | |
|---|---|---|
| | | (Required), and will be returned if userStoreType=AD. |
| serachContext | Optional | The tree construction of client AD system.It will not be returned if userStoreType=JDBC , and will be returned if userStoreType=AD. |
| groupDN | Optional | The user group of client AD system.It will not be returned if userStoreType=JDBC, and will be returned if userStoreType=AD. |
| userGroupTableName | Optional | The user and group relation table.It will be returned if userStoreType=JDBC, and will not be returned if userStoreType=AD. |
| userFK | Optional | The FK of user table.It will be returned if userStoreType=JDBC, and will not be returned if userStoreType=AD. |
| groupFK | Optional | The FK of group table.It will be returned if userStoreType=JDBC, and will not be returned if userStoreType=AD. |
| userSearchFilter | Optional | It will not be returned if userStoreType=JDBC, and will be returned if userStoreType=AD. |
| groupSearchFilter | Optional | It will not be returned if userStoreType=JDBC, and will be returned if userStoreType=AD. |
| userMemberOfAttributeName | Optional | It will not be returned if userStoreType=JDBC, and will be returned if userStoreType=AD. |
| dnAttributeName | Optional | It will not be returned if userStoreType=JDBC, and will be returned if userStoreType=AD. |
| userQuerySql | Optional | It will be returned if userStoreType=JDBC, and will not be returned if userStoreType=JDBC. |
| groupQuerySql | Optional | It will be returned if userStoreType=JDBC, and will not be returned if userStoreType=JDBC. |

JSON EXAMPLE:
userStoreType=JDBC:

```
{
 "status": "OK",
 "response": {
"userStoreType": "JDBC" ,
"dbType": "Oracle",
"url": "jdbc:mysql://172.16.10.189:3306/securlogin_clouddb",
"loginID": "administrator",
"password": "password22",
"userTableName": "userTable",
"userUID": "email",
"userAttributesMapping": "email=kehu_email;phoneNumber=kehu_phone;....
extAttribute=ext_name1,ext_name2",
"groupTableName": "groupTableName",
"groupUID": "groupID",
"groupAttributesMapping": "groupID=groupID1;groupName=groupName1",
"userQuerySql": "select * from user",
"groupQuerySql": "select * from group",
"reconSchedule": "0 0/1 * 1/1 * ? *"
}
}
```

userStoreType=AD:

```
{
 "status": "OK",
 "response": {
"userStoreType": "AD" ,
"url": "ldap://192.168.1.1:389",
"loginID": "administrator",
"password": "password22",
"userAttributesMapping": "email=kehu_email",
"useSSL": "false",
"ntDomain": "app-sg-isi",
"userSerachContext":"DC=app-sg-isi,DC=i-sprint,DC=com",
       "userSerachFilter":"{Object}",
"groupDN":"OU=ZH_PE,OU=ZH,OU=Users,OU=CN,DC=app-sg-isi,DC=i-sprint,DC=com",
"reconSchedule": "0 0/1 * 1/1 * ? *"
}
}
```

# Error Handling

Responses are formatted as a JSON object with a top-level status key.
Successful responses will have a status value of "OK" and a response key. The response will either be a single object or a sequence of other JSON types, depending on which endpoint is called.

```
{
 "status": "OK",
 "response": {
   "key": "value"
  }
}
```

Values are returned as strings unless otherwise documented.
Unsuccessful responses will have a status value of "FAIL", an integer code, and a message key that further describes the failure. A message_detail key may be present if additional information is available (like the specific parameter that caused the error).

```
{
 "status": "FAIL",
 "code": 40002,
 "message": "Invalid request parameters",
 "message_detail": "username"
}
```

The HTTP response code will be the first three digits of the more specific code found inside the JSON object. Each endpoint's documentation lists HTTP response codes it can return. Additionally, all API endpoints that require a signed request can return the following HTTP response codes:

| Code | Description |
|---|---|
| 200 | The request completed successfully. |
| 401 | The "Authorization", "Date", and/or "Content-Type" headers were missing or invalid. |
| 403 | This integration is not authorized for this endpoint or the ikey was created for a different integration type (for example, using an Auth API ikey with Admin API endpoints). |
| 404 | Not found. |
| 405 | The request's HTTP method is not valid for this endpoint (for example, POST when only GET is supported). |
| 413 | Request Entity Too Large. |
| 415 | Unsupported Media Type. |
| 429 | The account has made too many requests of this type recently. Try again later. |
| 500 | System Error. |
| 2001 | Signature/authentication failure. |
| 2002 | Invalid signature private key. |
| 2003 | Signature algorithm is not existing. |
| 2004 | Invalid or null signature request time. Please follow RFC 2822 format. |
| 2005 | Invalid or null HTTP method. |
| 2006 | The API path of request is required. |
| 2007 | Application key is required. |
| 2008 | Invalid application key. Failed to obtain secure key using application key. |
| 80001 | User ID is required. |
| 80004 | User ID is duplicate. |
| 80005 | User is not existing or duplicate ID. |
| 80006 | Generate OTP failed. |
| 80007 | Send SMS message failed. |
| 80009 | Application key is required. |
| 80012 | OTP verification failed. |
| 80015 | Authentication method cannot be null. |
| 80016 | Unknown authentication method. |
| 80019 | Voice message sent failed. |
| 80020 | Email message sent failed. |
| 80021 | User email is not existing. |
| 80034 | Push message sent failed. |

| | |
|---|---|
| 80035 | Policy of application cannot be null. |
| 80043 | Cannot find the tfa login status with txid or sessionid. |
| 80050 | Txid or sessionId cannot be empty. |
| 80057 | Config info of application cannot be null. |
| 80066 | Default global policy cannot be null. |
| 90001 | The request parameter error, please check. |
| 90002 | Txid is not existing. |
| 90003 | The email or phone number have been used. |
| 90004 | Have time out. |
| 90005 | All the 2fa method have been banned. |
| 90006 | This 2fa method have been banned. |
| 90007 | Mobile number or email is already in use. |