IDM - Identity Module

System Documentation

Overview

IDM is an authentication and authorization module built on top of AWS Cognito and offered as a service through an API interface. IDM enables users to securely authenticate access to multiple applications while logging in only once with just one set of credentials. Applications consuming IDM services can optionally maintain their own user database storing user metadata that does not belong to IDM.

IDM natively stores user credentials and supports user metadata through the use of IDM User Partitions or subscription to an SNS topic. IDM's major features are: Registration, Authentication, Authorization and User Partitions. User information is captured in a single database, using a single set of credentials but shared among various BCCSA resources.

Application Features

Applications

Applications are the main entity on IDM. Every application can be configured with multiple Application Tokens and Roles (at least one of each is required); Users and User Partitions (ACLs) are associated with application roles.

Application Tokens

An application that consumes IDM services must be issued an access token. IDM supports multiple tokens per application. Application tokens are used to generate an AppID token used to identify the application on every request to IDM. Access to IDM resources will be granted based on the validity of the token presented.

Tokens can be enabled or disabled individually, so if a code breach happens, that specific token be disabled or if abusive usage is detected in can also be throttled. For this reason, every application client should use its own Application Token. Example: Mobile application uses one, Web application uses other and API uses a third one.

Application Roles

Every application can have one or more roles. Roles are directly associated with one and only one application.

A role dictates what a user can be allowed to do within an application.

Roles have the following attributes:

- Registration enabled: This option enables self-registration of users into that role. If not enabled, only System administrators will be allowed to associate users with the role.
- MFA Required: Currently not implemented. This will force all users of that role to use MFA.
- Super role: When enabled, users associated with the role will be able to access data from all users of the application this role is part of. The default access for a super role is read/write, however, this can be restricted to read-only through another property. The default behaviour for a role is to grant users access to their own information only.
    - Read Only: This option is only considered when Super Role is enabled. If enabled, it will allow users from this role to read all information from users of that application but not write.
- Allow federation: Currently not implemented. This will allow users to login with other identity providers such as Facebook and Google.

Note:

- Changes on those preferences in the UI or via API, makes immediate effect on registration calls.
- Sometimes roles are referred to as claims in the IDM documentation.

Partition ACLs

Partitions ACLs are the relationship between a partition namespace, an application role and the access level (Read or Read/Write). That said, we create partition ACLs to grant authorization for a specific role into a namespace and what that role can do on that namespace.

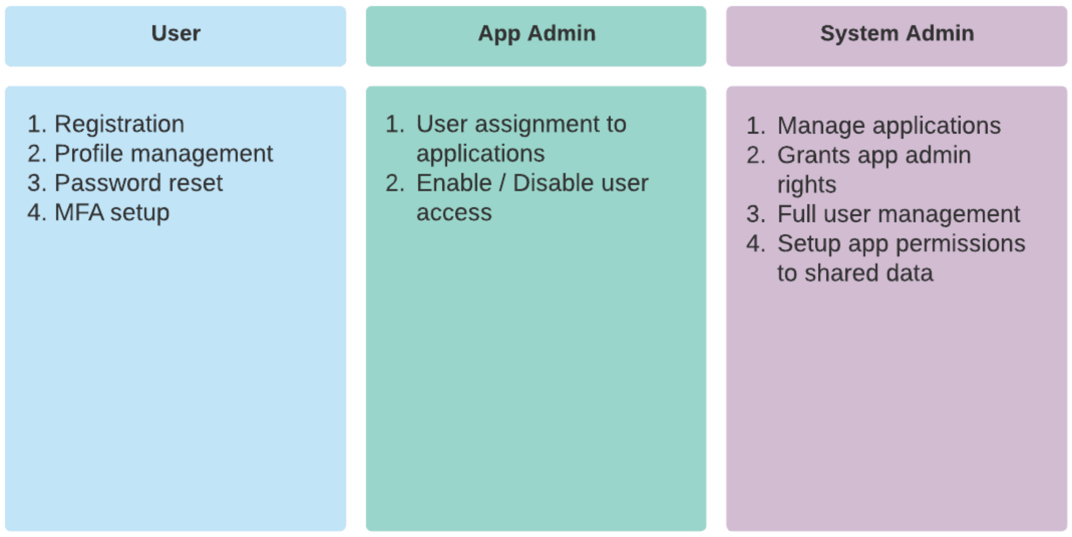
Namespaces are never pre-created to be used by ACLs, however the opposite is true, ACLs grant authorization to namespaces that will be created for each user on demand. So if we want to make usage of partitions system on IDM, we need to create an ACL for that namespace and role and then read/write it for each user.

Management Features

IDM UI

IDM UI is implemented via a small front end that is itself an IDM registered application. The IDM UI makes use of the same API calls available for all applications. It implements basic user, application and system management through a web interface. This UI also implements popup based authentication for other applications. This feature will not be discussed here in detail as that feature is being deprecated and highly discouraged for compatibility purposes.

IDM Roles



User	App Admin	System Admin
1. Registration 2. Profile management 3. Password reset 4. MFA setup	1. User assignment to applications 2. Enable / Disable user access	1. Manage applications 2. Grants app admin rights 3. Full user management 4. Setup app permissions to shared data

The following 3 roles were defined for the IDM application (UI and API) that are fundamental for the understanding of the system.

- **System Admin**: Users responsible for managing the IDM system. Allowed to create/delete applications and its roles & tokens, manage partitions ACLs, create/delete/list all users and its roles in all applications.
- **App Admin**: Users responsible and authorized to manage users associated with that specific application, also able to create users with the role from application they administrate.
- **Users**: Normal users are allowed to self-register, manage their own personal information and access applications they are part of.

Important Note: After authentication is processed, all user roles (or claims) are returned in the user token and are available to all applications consuming IDM services. It is technically possible for an application to decide to grant access to a user based on, for example, his/her association with the ***IDM User Admin*** role. However this is **extremely discouraged,** consumer applications should only consider their own roles when determining access or deciding what API calls to allow for a user.

IDM API

The IDM API is the only interface allowed to communicate with Cognito and the database storing user information. Therefore, all Authentication and Registration calls must be done directly with IDM via public API.

Documentation can be seen and tested **here**

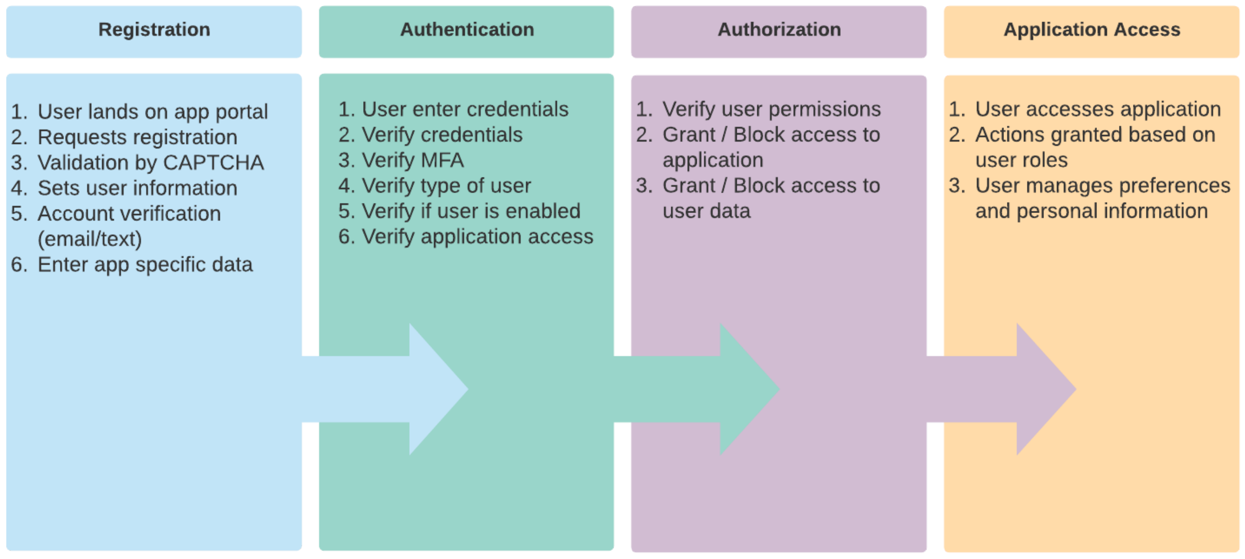
Notes:

- All API calls are done through secure protocol (HTTP/SSL) with valid certificate and data sent on plain-text, no base64 encoding.
- API communication format is `application/json` and no other format is accepted at the moment. Therefore, content-type and accept headers are not required but will be allowed if specified.
- Every API call must be identified with what application is making that call. This can be done by setting the `AppAuth` parameters on the query URL or header with the AppID token generated using the Application access token and Application access secret. Please, see more about AppID on its section at Application Features -> Application Tokens

- Most of the API calls require Authorization header to be set. Please, check API documentation to better understand what calls will need it. If not provided, a 401 status code will be returned.
- API errors are based on body code and not status code, so every error will return 4xx code and the body will contain further information. Please, check the Error model to understand error structure.
- 5xx status codes are not expected and should be reported when happened.
- Bugs and support with API calls **can't** be done **without** transaction ID, returned on every API call response. This is crucial for our team to identify the transaction and what happened. So please, make sure to include it when reporting a bug or asking for support.
- Headers & Query parameter keys are case insensitive.

User Features

The main user features are briefly illustrated on the flow below.


Registration
1. User lands on app portal
2. Requests registration
3. Validation by CAPTCHA
4. Sets user information
5. Account verification (email/text)
6. Enter app specific data
Authentication
1. User enter credentials
2. Verify credentials
3. Verify MFA
4. Verify type of user
5. Verify if user is enabled
6. Verify application access
Authorization
1. Verify user permissions
2. Grant / Block access to application
3. Grant / Block access to user data
Application Access
1. User accesses application
2. Actions granted based on user roles
3. User manages preferences and personal information

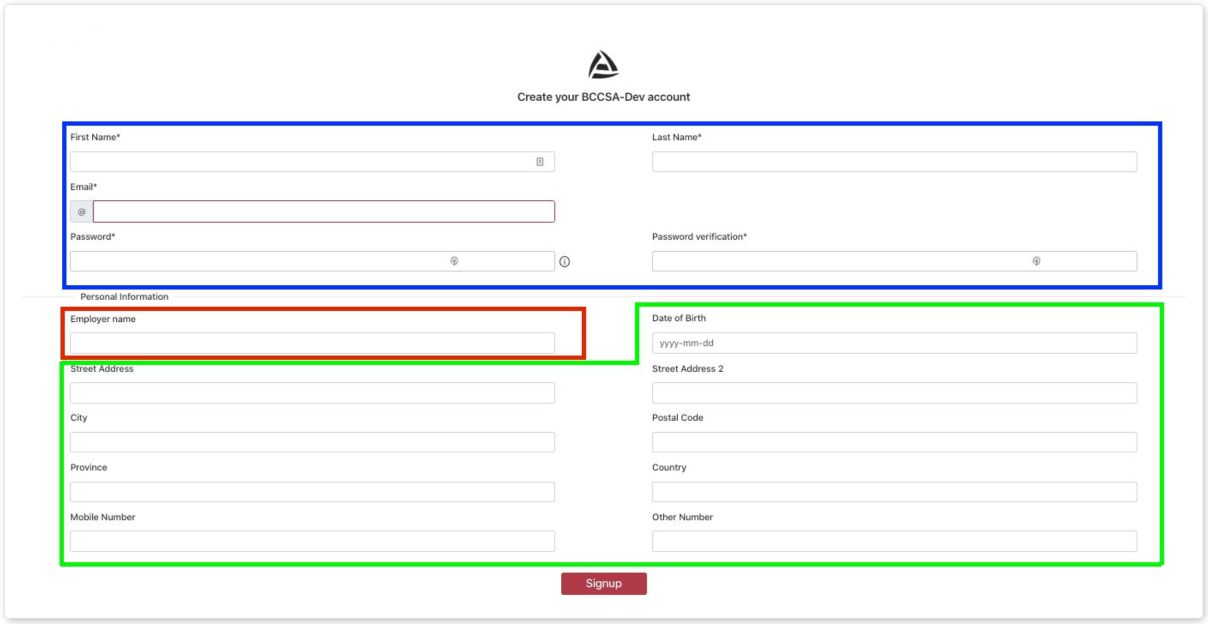
Registration

Currently, registration is solely done using the IDM UI but that functionality is exposed via API for applications that wish to make their own registration processes. The registration process implemented by

IDM creates an ID, confirms its address and also allows the calling application to determine what role(s) it should associate the new user with.

User registration involves gathering user information, sending a confirmation email to validate the user email (done by IDM), attaching user into specified roles (Role with registration enabled only), confirming it through IDM UI or API call and then launching the client application for the first time.

The existing process implemented on IDM UI collects basic and business related user information as required by BCCSA applications and stores it into user partitions. If the registration call succeeds, a registration token is returned by the API and when merged with the second registration code sent on the user email, can be used to confirm the account via API or IDM UI.


Create your BCCSA-Dev account

First Name*
Last Name*
Email*
Password*
Password verification*
Personal Information
Employer name
Date of Birth yyyy-mm-dd
Street Address
Street Address 2
City
Postal Code
Province
Country
Mobile Number
Other Number
Signup

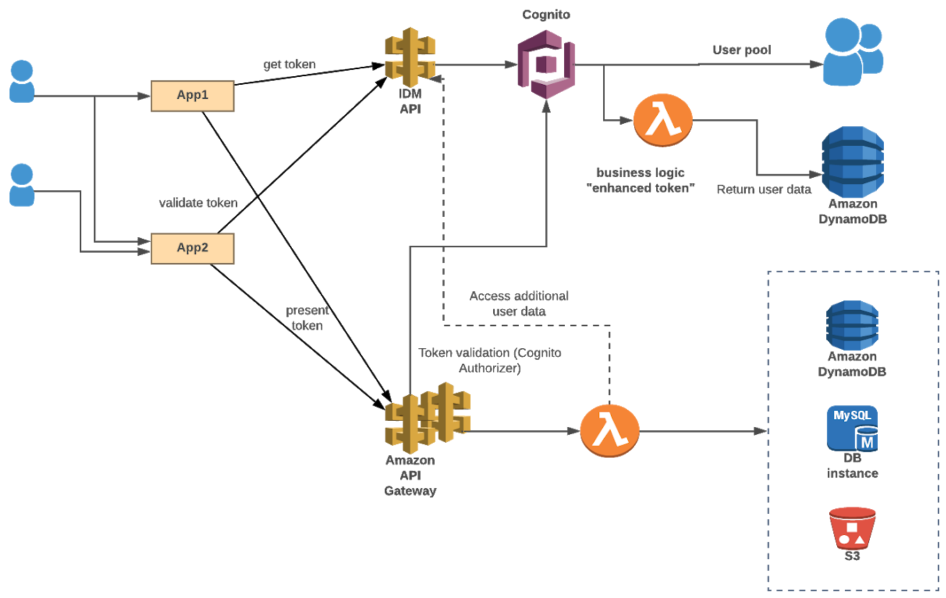
The screenshot above shows the information collected by IDM UI sectioned by where it will be saved. On the blue box we have the basic user information stored at user level and accessible by any application after the successful authentication. On the red box, we have professional information, stored on the user professional partition and accessible for all applications that have at least READ access into this partition. The green box indicates user personal information that is stored on the user personal partition and accessible for applications that have at least READ access to that partition.

Dummy account registration is encouraged to be done to better understand the registration process.

Authentication

Below is a flow describing 2 applications that are under the same apex domain and therefore share the authentication token. When **App1** authenticate with the IDM API it saves the token and other relevant information on the local cookie under `*.mydomain.com` to be later used with any other application that is served through any subdomain of the apex. When authorization is completed with success, IDM also returns the basic user information and all partitions that the caller application has access within the authorization token, so everything is done in one, compact call.

On this case, **App2** is launched after successfully authentication on **App1**, so it checks if the authorization token (from now on called IDM Token) is set, if so, it optionally calls IDM to validate that token or just opt to use it for authorizing the subsequent API calls on the application API. **App2** Will make use of the user information set by the **App1** and will not need any additional API calls to IDM.



Authorization

Following the authentication flow above, we are authenticated and now will make authorized API calls to the application API or IDM API with the IDM Token retrieved on the authentication call set as the `Authorization` header in the JWT format, `Authorization: Bearer <IDM Token>`.

When **App2** API is called, the authorization header is received by the API. API Gateway, if used, will take advantage of API Gateway Cognito Authorizer to validate the token with Cognito User Pool and decode it, launching a Lambda function with an already validated and decoded token, so the Lambda will just check the claims available on the token, to see if any of then are allowed to make the specified action on the API. If API Gateway is not used, the API will need to make the heavy lift of receiving the IDM Token, decoding it, validating it via Cognito User Pool public JWKS (available at `https://cognito-idp.REGION.amazonaws.com/USER_POOL_ID/.well-known/jwks.json`), checking if is not expired, and then authorizing the action with the available claims on the decoded token. Attention, JWKS is a very important step to avoid accepting IDM or Generic JWT tokens issued by other User Pool or attackers.

Please, refer to JWT documentation for further information.

User Partitions

User partitions can be defined as information buckets identified by a namespace and a user. They offer the flexibility of saving user information that may be useful for other applications on a fine-grained controlled namespace but also considered part of the user data. These partitions can be later read or updated by other applications, provided those applications have been granted access to that specific namespace.

Partitions have an encouraged limit of 200Kb and a hard limit of 390Kb. Attempts to write larger data sizes will not be accepted.

User Reflection

User reflection is an important IDM feature that guarantees that user information is synchronized between apps even when the user updates his information in one app but also exists on another app.

Whenever a user is updated via IDM API call, IDM will publish a message to a protected SNS Topic. Applications can subscribe to this topic to make sure its databases are up to date with the latest basic user information. For example, an application may want to cache locally the value of EmployerName.

Whenever a user record is updated, IDM will post a message containing the user record in json format (see sample at Appendix 1 - Sample update message.

Important Note: If SNS topic wasn't provided, please, request SNS User Reflection topic from your system administrator

ES6/NodeJS Module

Public registered on the official npm registry, `@ikonintegration/idmclient@1.x.x` is an ES6 nodejs module compatible with browser environment that abstracts most of the core logic describe here on this document and also shortcuts API calls and required parameters to consume IDM without much effort. It is strongly recommended that this module is used on the integration or at least the developer uses it as a base to make its own implementation.

Best security practices

- As stated before, API does not accept plan HTTP calls, therefore all HTTP+SSL calls should check against the API provided certificate and its authority. No self-signed certificates will be provided.
- It is encouraged that Application Access token and Application Access secret be stored in an obfuscated form on the front end and not visible by the client.
- Production Application tokens should never be stored on the application repository. Refer to AWS Secrets and inject them at build time for safer process.
- IDM Token should be stored on browser cookies that are restricted to known domains.
- Never store IDM Token at server sessions or at the same location Application tokens reside. This can lead to attacks where the attacker behaves like the registered application and can make actions on the user's behalf.
- Never load external resources from unknown resources, XSS vulnerability.
- User password should never be saved or stored, it should be transferred to API call and then cleared from memory.

Tokens

IDM Token (JWT)

IDM Token, here referred to as IDM Token, JWT and authentication token, is the token returned by the IDM API after successful authentication and then used as the authorization hash used on application and IDM API calls. This token can be decoded for retrieving very basic user information and its claims (roles) of all applications that user is associated with.

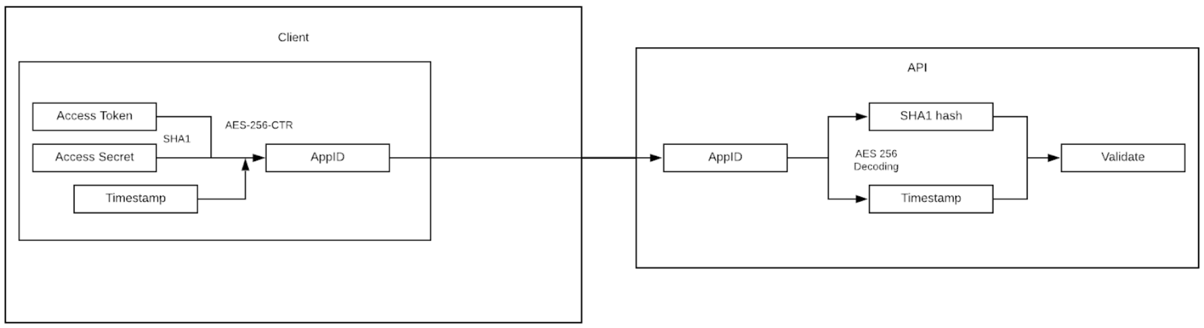
A token has a default validity of one hour. And when expired, renewal token, returned by the API authorization call should be used within validation API call to retrieve a new valid token.

Basic knowledge of JSON web token methodology is strongly encouraged before working with IDM.

App ID Token

AppID token is a generated token sent on every IDM API request to identify the application and validate its usage.

This token is sent in the header or query parameter `AppAuth` and calculated using the Application Access key and secret. The generated token includes the current unix timestamp and API allows tokens generated at maximum of 300 seconds before the request. Therefore, it should be calculated/generated at every request to avoid using expired tokens.



Above the token generation and validation process is briefly illustrated. For more detailed explanation, please, check below.

1-

- Access token and secret SHA1 is generated from the string `{“token”:”AppToken”, “secret”:”AppSecret”}` where AppToken and AppSecret is replaced with the actual keys. This first token is known as `verificationToken` and will be used on the next step.
- Another string is created from `{“token”:”verificationToken”,”timestamp”: 158267906300 }`. Where verificationToken is the token generated on the previous step and timestamp is the unix timestamp in **milliseconds** not in seconds as the usual timestamp formatting.
- Create AES-256-CTR of the string created on the previous step with a random 16 bytes initialization vector (IV) and salted with the key provided by the system administrator and referenced as AppID rotative key. If no key was provided, please, ask the system administrator.
- And finally but not least, format your hash as `IV:AES`. Where IV is your 16 bytes random initialization vector and AES is your previously generated AES hash. This is your **AppID** valid for the next 300 seconds.

Important Note: This algorithm is implemented on IDM ES6/Nodejs module on the class AppID, file AppID.js. Developers are encouraged to use the existing implementation or base their code into this existing class.

Setup

1. Define application name, what roles and what are their behaviour on the application, what partitions and informations will be saved on IDM and request it to be created on the development environment by your system administrator.
2. Application access token and secret, partitions namespaces, roles IDs, API endpoint, SNS Reflection Topic arn and admin account will be provided by you system administrator.

Appendix 1 - Sample update message

```
{
   "Records": [
      {
         "EventSource": "aws:sns",
         "EventVersion": "1.0",
         "EventSubscriptionArn":
"arn:aws:sns:ca-central-1:222275311114:IDM-API-dev-IDM-USER-UPDATE:d6c7aaa9-1404-46
08-b589-f0bda96d04ab",
         "Sns": {
            "Type": "Notification",
            "MessageId": "ce9a4ef2-691c-5111-aab2-1845f6f74b03",
            "TopicArn":
"arn:aws:sns:ca-central-1:222275311114:IDM-API-dev-IDM-USER-UPDATE",
            "Subject": null,
            "Message":
"{\"user\":{\"id\":\"6636a2fa-9f72-4c39-bc01-5c1a2abe9477\",\"firstName\":\"Jane\",\"lastName\":\
"Doe\",\"email\":\"jdoe@bccsa.ca\",\"isEnabled\":true,\"mfaEnabled\":false,\"objectClass\":\"user\"
,\"createdOn\":1579124697257,\"lastLogin\":1581095782818,\"lockedUntil\":-1,\"updatedOn\":15
79191867329,\"confirmationDate\":1579124726567,\"updatedBy\":\"6636a2fa-9f72-4c39-bc01-5
c1a2abe9477\",\"employer\":\"::EmptyEmployer::\",\"linkingRoles\":[\"1111f352-c7f_CCPO\",\"aa
aabbd-bbbb-427e-9c92-afd74ab19431\"],\"transactionID\":\"c0c97f13-83dc-4700-ad15-3127b04
d8558\",\"parts\":{\"ca.bccsa.personal\":{\"value\":{\"driverLicense\":\"0909090\",\"dateOfBirth\":\"
1993-09-17\",\"address\":{\"country\":\"Canada\",\"province\":\"BC\",\"city\":\"New
Westminster\",\"street\":[\"625 Agnes Street\",null],\"postalCode\":\"V3M
5Y4\"},\"mobilePhone\":\"6049999999\"}},\"ca.bccsa.professional\":{\"value\":{\"employer\":{\"nam
e\":\"BCCSA\"}}}}},\"eventType\":\"N_UPDATE\"}",
            "Timestamp": "2020-02-07T17:26:03.019Z",
            "SignatureVersion": "1",
            "Signature":
"eJcLZy6iI73pdciVORBAnrx7oZ3XQKabr3TnPphFYvOINwJh9/7yIG6Zw+AfvSewxyI5UA2srx2X
xmVZWkbFI0j3r2bAga1YXM5+i5QQ8c4AS1NVnK548AKIKs90C8F58nNaFJgmsN8Q4Lm1QUu
R1IX7Uglev9uDYu6qnbrBPHHr7/xqDLmIiTAIHENNUep+I9yyyYv8chbst1qPRYvYjUvWs9WrXbr
rW+iA5bnjxkPInv1OYzQCmhfOIDeGJFD+fCi6nqZLPIOZI/fKHmIy3x1p5IsSnjzB7syN7JIxeZE9M
Q3nIB6ARCr56Wvxx6VCK/MvYTegjIZ0xDUrUkotIg==",
            "SigningCertUrl":
"https://sns.ca-central-1.amazonaws.com/SimpleNotificationService-a86cb10b4e1f29c941702d
737128f7b6.pem",
            "UnsubscribeUrl":
"https://sns.ca-central-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:c
```

a-central-1:222275311114:IDM-API-dev-IDM-USER-UPDATE:d6c7aaa9-1404-4608-b589-f0bd
a96d04ab",
            "MessageAttributes": {}
         }
      }
   ]
}

The actual message payload is

{
  "user":{
    "id":"6636a2fa-9f72-4c39-bc01-5c1a2abe9477",
    "firstName":"Jane",
    "lastName":"Doe",
    "email":"jdoe@bccsa.ca",
    "isEnabled":true,
    "mfaEnabled":false,
    "objectClass":"user",
    "createdOn":1579124697257,
    "lastLogin":1581095782818,
    "lockedUntil":-1,
    "updatedOn":1579191867329,
    "confirmationDate":1579124726567,
    "updatedBy":"6636a2fa-9f72-4c39-bc01-5c1a2abe9477",
    "employer":"::EmptyEmployer::",
    "linkingRoles":[
      "1111f352-c7f_CCPO",
      "aaaabbd-bbbb-427e-9c92-afd74ab19431"
    ],
    "transactionID":"c0c97f13-83dc-4700-ad15-3127b04d8558",
    "parts":{
      "ca.bccsa.personal":{
        "value":{
          "driverLicense":"0909090",
          "dateOfBirth":"1993-09-17",
          "address":{
            "country":"Canada",
            "province":"BC",
            "city":"New Westminster",
            "street":[
              "625 Agnes Street",
              null
            ],

```
            "postalCode":"V3M 5Y4"
          },
          "mobilePhone":"6049999999"
        }
      },
      "ca.bccsa.professional":{
        "value":{
          "employer":{
            "name":"BCCSA"
          }
        }
      }
    }
  },
  "eventType":"N_UPDATE"
}
```