Alchemer Mobile Customer Authentication

> Customer Authentication on legacy SDKs (earlier than v6.0) is no longer actively supported. We encourage you to update to SDK v6.5 if using the customer authentication feature.

What is Customer Authentication?

Customer Authentication is an optional feature introduced in the iOS and Android legacy SDKs. It is now only supported in the new SDKs, enabling multiple customers to use your app on a single device while ensuring the separation and security of their data. Once enabled, customers will have access to their personal set of interactions, engagement records, and a Message Center. Additionally, their data will be encrypted both within the SDK and during transmission. If you choose not to enable Customer Authentication, the existing behavior will persist, where all Alchemer Mobile information is shared among all potential users of the device.

To use Customer Authentication, you must:

- Have an authentication system you control that allows multiple people to log into your app.
- Generate JWT tokens in that authentication system and return them to the app.
- Update your app code to call the appropriate Alchemer Mobile login method with the generated JWT.

Below we will walk through the requirements for the JWT token and sample code of the work required in addition to integrating the SDK.

If you do not plan to use the new Customer Authentication feature, this work is not required.

Authentication Flow

1. Customer enters login information in your app’s login form.

2. Your server processes that authentication information and finds the person has supplied valid auth information. It generates a JWT for that user and returns it to the device.

3. You must pass the JWT to the platform-specific “login” method to create/fetch the logged in Alchemer Mobile (Apptentive) Conversation.

4. The Alchemer Mobile (Apptentive) SDK sends the JWT to our server, if valid, it unlocks the encrypted Conversation on the device.

5. The callback you passed into the Alchemer Mobile (Apptentive) login API method is called with the status of the login.

The JWT Token

To authenticate requests made to the Alchemer Digital API, SDK 6.5 and above login with JWT token. To establish this authentication, your server is responsible for generating the JWT token as part of your user authentication process.

Libraries

jwt.io has an extensive list of libraries that can be used to work with these tokens.

Requirements

To generate a JWT token, there are three required pieces:

- Payload
  - See the required fields below.
- Signing secret
  - This can be found on the API & Development page, and should be treated as raw UTF-8 data (rather than base-64 or hexadecimal).
- Signing algorithm
  - Alchemer Mobile (Apptentive) expects tokens to be signed with the **HS512** algorithm.

Field Descriptions

The payload of a JWT is an arbitrary hash of keys ("claims" in JWT parlance) and values; however, Alchemer Mobile (Apptentive) does require a few claims to generate a valid token.

- iat – ("issued at") **REQUIRED**
  - The UTC timestamp when the token was generated
- exp – ("expiration") **OPTIONAL**
  - The UTC timestamp to invalidate the token
  - Set this to be slightly longer than your authentication service's session length
  - Alchemer Mobile (Apptentive) enforces a maximum token length of 30 days
- sub – ("subject") **REQUIRED**
  - A unique, immutable identifier for the user used for database lookup
    - This value must not change, otherwise Alchemer Mobile (Apptentive) will treat it as a new Conversation and the customer will lose their message history.
  - Uniqueness for this field is scoped to each App in Alchemer Mobile

In Ruby, the hash payload would look like this:

```
{
  iat: 1501548760,
  exp: 1501807985,
  sub: "unique_immutable_value_for_user123121"
}
```

Generating a JWT

```
payload = {
  iat: 1501548760,
  exp: 1501807985,
  sub: "unique_immutable_value_for_user123121"
}
token = JWT.encode(payload, apptentive_provided_secret, "HS512")
```

The resulting token will be 3 base64-encoded sections separated by periods looking something like this:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1bmlxdWVfaW1tdXRhYmxlX3ZhbHVlX2Zvcl91c2VyMTIzMTIxIiwiaW
F0IjoxMjMxMjMxMjMxLCJleHAiOjEyMzEyMzEyMzJ9.1xUmdJqFygoDI-1N2gJzICvC-qdaWUATBBUM35dxrPM
```

Example Code

Two very basic apps (written in Ruby and Java) are provided in the git repository below to show an example of an API you might create to authenticate a user and generate a JWT token. Note that the details of user authentication and unique user token generation are skimmed over and your team must implement them.

https://github.com/apptentive/apptentive-sdk-auth-example

Related Articles