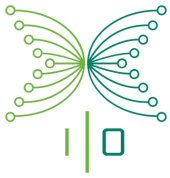
# Backend - Authentication



# Introduction

This document explains the authentication process between the sales app and the backend, in every request and response. As well as the process needed to generate the tokens for that interaction.

# Applies for

- Backend: Version > 1.0.0
- Sales App: Release > 1.15.0

# Authentication

## Requests

The authentication process from a sales app instance to the Back-end is made following the **Bearer Token Protocol** defined in RFC6750[1]. In our case, tokens are statically generated, no generation endpoint is available.

Essentially, every sensitive request will need the Header:

| | |
|---|---|
| Authorization | Bearer {TOKEN} |

There are 3 types of endpoint access:

- **Public**: No token is required (example: /healthcheck or bitgo callback )
- **Private ReadOnly**: This endpoints will required a **Read Only Token** or **Read-Write Token**: Returns error if not defined or incorrect. (example: GET -> /price/rates)
- **Private ReadWrite**: Only **Read-Write Token**, will work. (example POST -> /invoiceWallet)

[1] https://tools.ietf.org/html/rfc6750

### Responses

Sometimes, it's the backend the one that initiates the interaction with the sales app, and in particular cases there is sensitive information sent back in the payload response. In these cases, the payload needs to be encrypted with a special key (SIGNATURE) associated with that client TOKEN, and include the client ID in the Header under the Header Key : **"X-Client-Id"**.

The encryption will need to follow the standard defined for the **Stanford Javascript Crypto Library (SJCL).**

## Revoke Access

To revoke access to a particular token, it's as simple as deleting the entry from the "access_token" table.

## Security Concerns

If the database gets compromised by enabling someone read-write permissions, they won't be able to compromise any access, only a deny of service is possible. Detail:

- They won't be able to read a token, as only a hash is stored
- Any new "*fake*" entry entered won't work as token, even if the same hash function was applied to it, as the payload is encrypted with the server deploy key. Even if they copy another registry encrypted data, that won't work because it's salted.
- Altering an existing entry would invalidate it, for the same reasons.

## Access Token Generation

The Access Token is composed of the following elements:

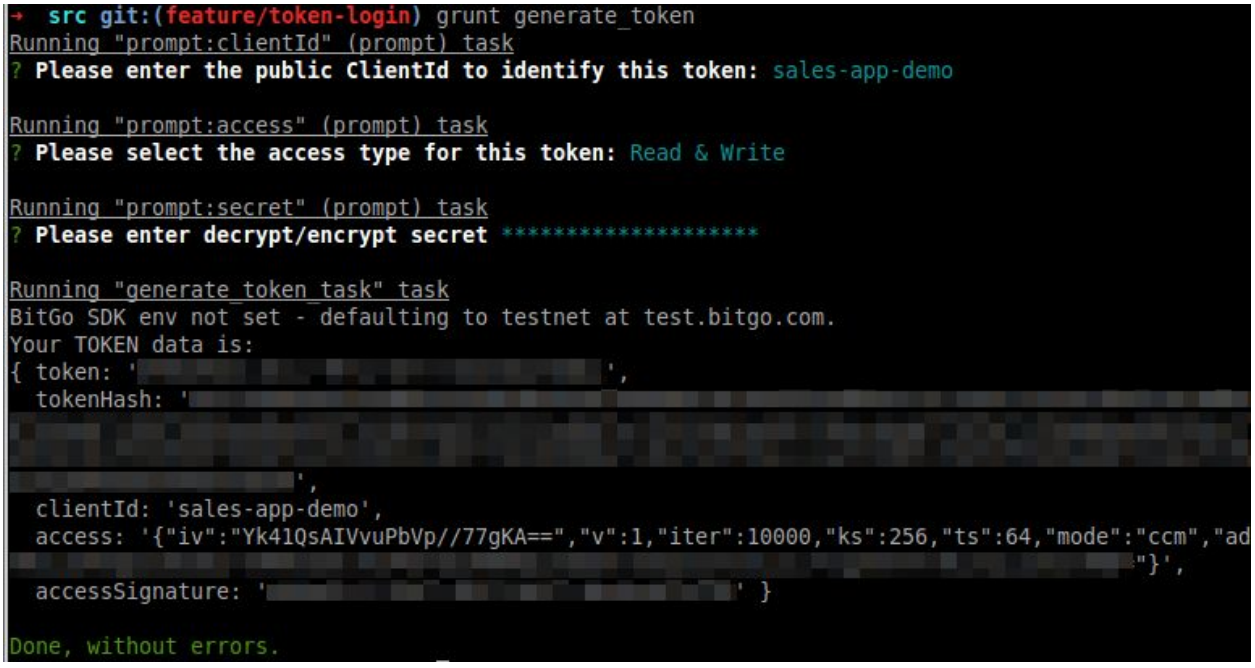
- **Token**: The TOKEN itself **[Sensitive]**
- **Token Hash**: A salted hash of the token, this needs to be stored in the Database
- **Client ID**: A descriptive unique id of the token client. (example: "Sales-App-JPN", "Atix-Read-Only" )
- **Token Access Type**: { R=ReadOnly, RW: ReadWrite }
- **Access Signature**: The signature associated with this Token, needs to encrypt payload responses **[Sensitive]**

- **Access Payload**: Encrypted Salted Payload Data, containing the Access Type and the Signature

The Token generation process can be performed by a *grunt script* ("generate_token") created for this purpose. It will ask for the application secret[2], the client-id and the access type, and will generate a completely new Access Token item. Which will then be needed to be entered into the 'access_token' table in the corresponding database *{token hash, client-id and access payload}*.

**IMPORTANT**: *The data will be returned by the script as text console output, so it's extremely important to run this script on a secured environment, the **token & Access Signature** must be kept safe after the process.*

## Example:




## Backend Configuration

- Insert the new Access Token data into the database table "access_data" *{token: "token_hash", client_id : "client_id", access: "access payload" }*

[2] The secret value asked as "Please enter encrypt/decrypt secret can be found in development.json encrypted file as "secret"

## Sales-App Configuration

- Set up the Backend **TOKEN**
- Set up the Backend **ClientId**
- Set up the **Backend Access Signature**

These are currently plain text settings, but would need to be encrypted and turned unreachable from repository access.