

Encrypts sensitive login secrets stored in the database such as a login service's application secret key and users' access tokens.

Generating a Key

The encryption key is 16 bytes, encoded in Base64.

To generate a key:

```
$ meteor node -e 'console.log(require("crypto").randomBytes(16).toString("base64"))'
```

Using oauth-encryption with accounts

On the server only, use the `oauthSecretKey` option to `Accounts.config`:

```
Accounts.config({ oauthSecretKey: 'onsqJ+1e4iGF1V0nhZYobg==' });
```

This call to `Accounts.config` should be made at load time (place at the top level of your source file), not called from inside of a `Meteor.startup` block.

To avoid storing the secret key in your application's source code, you can use [Meteor.settings](#):

```
Accounts.config({ oauthSecretKey: Meteor.settings.oauthSecretKey });
```

Migrating unencrypted user tokens

This example for Twitter shows how existing unencrypted user tokens can be encrypted. The query finds user documents which have a Twitter access token but not the `algorithm` field which is created when the token is encrypted. The relevant fields in the service data are then encrypted.

```
const cursor = Meteor.users.find({
  $and: [
    { 'services.twitter.accessToken': { $exists: true } },
    { 'services.twitter.accessToken.algorithm': { $exists: false } }
  ]
});

cursor.forEach((userDoc) => {
  const set = {};

  ['accessToken', 'accessTokenSecret', 'refreshToken'].forEach((field) => {
    const plaintext = userDoc.services.twitter[field];

    if (!_.isString(plaintext)) {
      return;
    }

    set[`services.twitter.${field}`] = OAuthEncryption.seal(
      plaintext,
```

```
        userDoc._id
      );
    });

    Meteor.users.update(userDoc._id, { $set: set });
  });
```

Using oauth-encryption without accounts

If you're using the oauth packages directly instead of through the Meteor accounts packages, you can load the OAuth encryption key directly using `OAuthEncryption.loadKey` :

```
OAuthEncryption.loadKey('onsqJ+1e4iGF1V0nhZYobg==');
```

If you call `retrieveCredential` (such as `Twitter.retrieveCredential`) as part of your process, you'll find when using oauth-encryption that the sensitive service data fields will be encrypted.

You can decrypt them using `OAuth.openSecrets` :

```
const credentials = Twitter.retrieveCredential(token);
const serviceData = OAuth.openSecrets(credentials.serviceData);
```

Using oauth-encryption on Windows

This package depends on [npm-node-aes-gcm](#), which requires you to have OpenSSL installed on your system to run. To install OpenSSL on Windows, use one of the binaries on [this page](#). Don't forget to install the Visual Studio 2008 redistributables if you don't have them yet.