



class Auth::SCRAM

Authentication using SCRAM

Table of Contents

- 1 [Synopsis](#)
- 2 [Methods](#)
 - 2.1 [new](#)
 - 2.2 [derive-key](#)
 - 2.3 [client-key](#)
 - 2.4 [stored-key](#)
 - 2.5 [client-signature](#)
 - 2.6 [server-key](#)
 - 2.7 [XOR](#)
 - 2.8 [normalize](#)
 - 2.9 [encode-name](#)
 - 2.10 [decode-name](#)
 - 2.11 [test-methods](#)

```
unit package Auth;  
class SCRAM { ... }
```

Synopsis

```
# User defined class with the task of communicating with a server  
# for the authenticating process  
class MyClient {  
  
    submethod BUILD {  
        # Establish server connection  
    }  
  
    method client-first ( Str:D $client-first-message --> Str ) {  
        # Send $client-first-message to server and return server  
        # response as server first message  
    }  
  
    method client-final ( Str:D $client-final-message --> Str ) {  
        # Send $client-final-message to server and return server  
        # response as server final message  
    }  
  
    method error ( Str:D $message --> Str ) {  
        # Errors? nah ... (Famous last words!)  
    }  
}
```

```
# Initialize SCRAM with above class
my Auth::SCRAM $sc .= new(
    :username<user>,
    :password<pencil>,
    :helper-object (MyClient.new),
);

my Str $error = $sc.start-scrum;
```

While a client authentication is shown above Auth::SCRAM itself does not do any of this. The user of the Auth::SCRAM class provides a client-side or a server-side object with which the initialization phase will upgrade the object to a Auth::SCRAM object with a Auth::SCRAM::Client or Auth::SCRAM::Server role.

Specific details are shown in the pods for these role modules.

Methods

Auth::SCRAM has some methods which are mostly used by the client or server roles and have not much use by the caller directly and are therefore not explained.

new

Defined as

```
submethod BUILD (
    Str :$username,
    Str :$password,
    Str :$authzid,
    Bool :$case-preserved-profile = True,

    Callable :$CGH = &sha1,
    :$helper-object,
    Bool :$client-helper = True,
)
```

Initialize the process. The Cryptographic Hash function \$CGH is by default set to &sha1 from the OpenSSL::Digest module. The authorization id(\$authzid) is needed when you want things done with the privileges of someone else. The \$helper-object object is an object performing client or server side tasks. The methods in this object are called by the SCRAM methods. Only one of the client or server objects can be defined. The boolean \$client-helper specifies which type of object is provided.

The username and password are required when client side operations are performed. username, password and authorization id is not needed when a server side object is given because it will be provided by the client via an account registration mechanism and the clients first message will provide the username and authorization id to work with.

This method will take on the Client or Server role after inspecting the \$client-helper boolean.

Usernames and password (and maybe the authorization id) must be normalized. Older versions of the

scram process made use of SASLprep. This module will use the PRECIS framework defined by rfc7564 and crystalized in module Unicode::PRECIS. There are several classes and profiles in that framework. For usernames there is a profile to map case to lowercase and one to preserve it. This is controlled by the boolean \$case-preserved-profile and by default set to True. There is only one type of profile for passwords so no control needed there.

For specific client and server information look for the roles SCRAM::Client and SCRAM::Server. The rest of the methods are explained here but are only to be used by the afore mentioned roles.

derive-key

Defined as

```
method derive-key (
  Str:D :$username is copy, Str:D :$password is copy,
  Str :$authzid, Bool :$enforce = False,
  Buf:D :$salt, Int:D :$iter,
  Any:D :$helper-object
--> Buf
)
```

Calculate the derived key from the password, salt and number of iterations. The cryptographic hash function is selected or provided at the instantiation phase.

The username and password are normalized using the PRECIS framework described above before calculating. Furthermore there are two procedures which can be followed. a) preparation and b) enforcement. Preparation is mostly done at the client side and may modify the original string The enforcement does the same but add some extra tests before accepting the string. Enforcement mostly takes place at the server. This is selectable with \$enforce which is by default False.

When the method mangle-password is defined in the user provided helper object, that method will then be called. The signature can be something like the following;

```
method mangle-password (
  Str :$username,
  Str :$password,
  Str :$authzid
--> Buf
)
```

When the method is not defined, the following default action takes place;

```
my Buf $mangled-password .= new($password.encode);
```

client-key

Defined as

```
method client-key ( Buf $salted-password --> Buf ) {
```

See rfc5802

[stored-key](#)

Defined as

```
method stored-key ( Buf $client-key --> Buf ) {
```

See rfc5802

[client-signature](#)

Defined as

```
method client-signature ( Buf $stored-key, Str $auth-message --> Buf ) {
```

See rfc5802

[server-key](#)

Defined as

```
method server-signature ( Buf $server-key, Str $auth-message --> Buf ) {
```

See rfc5802

[XOR](#)

Defined as

```
method XOR ( Buf $x1, Buf $x2 --> Buf ) {
```

Perform XOR operation on two buffers returning the result of it.

[normalize](#)

Defined as

```
method normalize (  
  Str:D $text, Bool:D :$prep-username!, :$enforce = False  
  --> Str  
)
```

Normalize \$text using \$case-preserved-profile boolean (described with new()), \$prep-username and \$enforce. \$prep-username boolean decides if a username or password profile is choosen. \$case-preserved-profile selects between one of two username profiles if \$prep-username is True. \$enforce selects the normalization procedure prepare or enforce.

[encode-name](#)

Defined as

```
method encode-name ( Str $name is copy --> Str ) {
```

Username must be encoded to protect the string format against use of some characters. These are ',' and '='. They must be translated to '=2C' and '=3D' resp.

[decode-name](#)

Defined as

```
method decode-name ( Str $name is copy --> Str ) {
```

This is the reversed process of encode-name.

[test-methods](#)

Defined as

```
method test-methods ( $obj, @methods --> Bool ) {
```

Method to check the provided user helper object for required methods