class Auth::SCRAM

Authentication using SCRAM

Table of Contents


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

Synopsis

See documentation of Auth::SCRAM::Client and Auth::SCRAM::Server.

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

Client side BUILD is defined as

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

  Callable :$CGH = &sha1,
  :$client-object!,
)
```

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 using the privileges of someone else. The $client-object object is an object performing client side tasks. The methods in this object are called by the methods in the SCRAM::Client Role.

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.

Server side BUILD is defined as

```
multi submethod BUILD (

  Bool :$case-preserved-profile = True,
  Callable :$CGH = &sha1,
  :$server-object!,
)
```

The $server-object object is an object performing server side tasks. The methods in this object are called by the methods in the SCRAM::Server Role. Username, password and authorization id are 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.

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

```
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

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

See rfc5802

stored-key

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

See rfc5802

client-signature

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

See rfc5802

server-key

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

See rfc5802

XOR

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

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

normalize

```
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

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

Usernames 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

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

This is the reversed process of encode-name.

test-methods

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

Method to check the provided user helper object for required methods