# role Auth::SCRAM::Server

Server side authentication using SCRAM

## Table of Contents


1 Synopsis

2 Read and writable attributes

2.1 client nonce

2.2 server nonce

2.3 extension data

3 Methods

3.1 generate-user-credentials

3.2 start-scram


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

# Synopsis

To do authentication on a server is a more complex process. The server needs to have some sort of user database to store the information in whiteout having the risk that passwords are stolen. The server role helps to convert the username and password into a credential tupple which can be stored without the risk that the password can be retrieved from this data. Below is a simple implementation of the necessary items. The methods get-the-users-database() and generate-a-proper-salt() are not implemented here.

```
class Credentials {

  has Hash $!credentials-db;
  has Auth::SCRAM $!scram handles <start-scram>;

```

That might be the initial setup. Now lets initialize ...

```
  # A user credentials database used to store added users to the system
  # Credentials must be read from somewhere and saved to the same somewhere.
  submethod BUILD ( ) {

    $!scram .= new(:server-object(self));
    $!credentials-db = self!get-the-users-database();
  }
```

Then there is a need to add a user. This method will ask the server role to generate the tupple and store it in the database.

```
  method add-user ( $username is copy, $password is copy ) {

    my Buf $salt .= self!generate-a-proper-salt();
    for $!scram.generate-user-credentials(
      :$username, :$password, :$salt, :iter(4096), :server-object(self)
    ) -> $u, %h {

      $!credentials-db{$u} = %h;
    }
  }
```

The rest of the class is needed in the authentication process

```
  method credentials ( Str $username, Str $authzid --> Hash ) {

    return $!credentials-db{$username};
  }

  # return server first message to client, then receive and
  # return client final response
  method server-first ( Str:D $server-first-message --> Str ) {

    is $server-first-message,
       'r=fyko+d2lbbFgONRv9qkxdawL3rfcNHYJY1ZVvWVs7j,s=QSXCR+Q6sek8bf92,i=4096',
       $server-first-message;

    < c=biws
      r=fyko+d2lbbFgONRv9qkxdawL3rfcNHYJY1ZVvWVs7j
      p=v0X8v3Bz2T0CJGbJQyF0X+HI4Ts=
    >.join(',');
  }

  # return server final message
  method server-final ( Str:D $server-final-message --> Str ) {

    is $server-final-message,
       'v=rmF9pqV8S7suAoZWja4dJRkFsKQ=',
       $server-final-message;

    '';
  }

  method error ( Str:D $message ) {

  }
}
```

Then initialize the server. Suppose the server knows about only two things, adding a user and authenticating a user. This will look someething like the code below. Once the idea is understood, more useful commands can be added.

```
# Server actions in advance ...
# - set up shop
my Credentials $crd .= new;

# - set up socket
# - listen to socket and wait
# - input from client
# - fork process, parent returns to listening on socket
# - child processes input for commands

# - command: add a user
# receive the password and password preferable over a secure connection
my Str $user = '...';
my Str $password = '...'
$crd.add-user( $user, $password);

# - command: autenticate
# receive client first message from client
my Str $client-first-message = "...";
my Str $emsg = $crd.start-scram(:$client-first-message);
```

# Read and writable attributes

These attributes must be set before scram authentication is started.

## client nonce

```
has Int $.c-nonce-size is rw = 24;
has Str $.c-nonce is rw;
```

Define a nonce. The result must be a hexadecimal string of the proper size generated by a base64 encoding operation. When not set, the class will makeup one of the default length of 24 octets and encode in base64.

```
$!c-nonce = encode-base64(
  Buf.new((for ^$!c-nonce-size { (rand * 256).Int })),
  :str
);
```

Normally it will not be needed to make one yourself. It was made available for testing purposes.

## server nonce

The server nonce can be modified too in the same way. Also for these attributes, it is not needed to use them.

```
has Int $.s-nonce-size is rw = 18;
has Str $.s-nonce is rw;
```

The default is generated like so.

```
$!s-nonce = encode-base64(
  Buf.new((for ^$!s-nonce-size { (rand * 256).Int })),
  :str
);
```

## extension data

```
has Str $.reserved-mext is rw;
has Hash $.extensions is rw = %();
```

These variables are not yet used in this module.

# Methods

## generate-user-credentials

```
method generate-user-credentials (
  Str :$username, Str :$password,
  Buf :$salt, Int :$iter,
  Any :$server-object

  --> List
)
```

When adding a user to the database this method helps creating the credential data needed to authenticate a user later. The method returns the normalized username(enforced profile) and a hash with the keys;

- **iter**. Number of iterations needed to generate the derived key
- **salt**. Salt used to generate that key. Both iter and salt are provided by the user
- **stored-key**. See rfc5802.
- **server-key**. See rfc5802.

When the method mangle-password is defined, that method will be called for this process.

## start-scram

```
method start-scram( Str:D :$client-first-message --> Str )
```

Start authentication. The authentication process is started by the user, so the server must receive the first client message before starting the scram process.

The calls to the user provided client object are as follows;

- **server-first**. After processing the client first message, the server first message must be send back to the client. As a result, the method must return the client final message.

```
method server-first ( Str:D $server-first-message --> Str )
```

- **server-final**. Purpose is to send the server final message to the client. If there is an error, return the error message. If successfull, return the empty string.

```
method server-final ( Str:D $server-final-message --> Str )
```

- **credentials**. This method is called several times to get user information. The method must return the same data returned from generate-user-credentials.

```
method credentials ( Str $username --> Hash )
```

- **mext**. Not yet implemented.
- **extension**. Not yet implemented.
- **authzid**. Not yet implemented.
- **error**. The error method is called with a message whenever there is something wrong. The message is prefixed wit 'e=' and must be send back to the client. The procedure is then terminated after returning from the error method and the procedure will return the same message to the caller of start-scram.

```
method error ( Str:D $message )
```

- **cleanup**. This optional method is called when all ends successfully. After returning from this method it returns an empty string('') to the caller of start-scram

```
method cleanup()
```