ACME Mangement Server

Release 0.3.1

December 13, 2016

Contents

Contents:

CHAPTER 1

Develop on ACMEMS

1.1 Manager

class acmems.manager.**ACMEManager**(*config*, *connect=True*)

Bases: object

ACME manager - high level ACME client; process authorizations via http01 automatically.

Variables

- **responses** (*dict*) – Responses to deliver; designed as answers for authorization challenges. dict[host][path] = value
- **authzrs** (*dict*) – List of current active acme.messages.AuthorizationResource
- ***config*** (*acmems.config.Configuration*) – Active configuration

log(**args*)

log something

connect()

initialize/setup ourself; load private key, create ACME client and refresh our registration

Raises

- ***acmems.exceptions.AccountError*** – could not load account
- ***acmems.exceptions.NeedToAgreeToTOS*** – terms of service are not accepted - cannot operate

load_private_key()

load our private key / the key to identify ourself against the ACME server. This key MUST NOT be used for certificates.

Raises ***acmems.exceptions.AccountError*** – something is broken with our account (mustly key not found)

create_private_key(*force=False*, *key_size=4096*)

create new private key to be used for identify ourself against the ACME server

Key is afterwards read via *load_private_key*!

Parameters

- **force** (*bool*) – create new key even key exists already
- **key_size** (*int*) – private key size in bits (at least 2048)

Raises ***acmems.exceptions.AccountError*** – account dir not found or private key will not be overriden (force is `False`).

init_client()
create ACME client

acquire_domain_validations(*validator*, *domains*)
requests for all given domains domain validations If we have cached a valid challenge return this. Expired challenges will clear automatically; invalided challenges will not.

Parameters **domains** (list of `str`) – List of domains to validate

Returns Challenges for the requested domains

Return type acme.messages.ChallengeBody

evaluate_domain_authorization(*authzr*, *validator*, *refresh_timer=None*)
Processes a given AuthorizationResource that was fetch from the authzrs cache or updated by *refresh_domain_authorization* / `acme.client.Client.poll`.

Renew revoked or expired ones. Refresh pending/processing authorizations

Parameters **authzr** (*acme.messages.AuthorizationResource*) – the authzr in question

Returns a valid authzr

Return type acme.messages.AuthorizationResource

Raises

- ***acmems.exceptions.AuthorizationNotYetProcessed*** – We have to wait while the ACME server processes the autzr
- ***acmems.exceptions.AuthorizationNotYetRequested*** – new authzr created; have to wait until someone requests it
- ***acmems.exceptions.ChallengesUnknownStatus*** – unknown status
- ***acmems.exceptions.NoChallengeMethodsSupported*** – HTTP01 is not supported
- ***acmems.exceptions.ChallengeFailed*** – challenge failed

refresh_domain_authorization(*validator*, *domain*)
Refreshes a authorization for status changes

Parameters **domain** (*str*) – domain name for the authorization

Returns a valid authzr

Return type acme.messages.AuthorizationResource

Raises

- ***acmems.exceptions.AuthorizationNotYetProcessed*** – We have to wait while the ACME server processes the autzr
- ***acmems.exceptions.AuthorizationNotYetRequested*** – new authzr created; have to wait until someone requests it
- ***acmems.exceptions.ChallengesUnknownStatus*** – unknown status
- ***acmems.exceptions.NoChallengeMethodsSupported*** – HTTP01 is not supported

new_domain_authorization(*validator*, *domain*)
Requests a complete new authorization for the given domain

Parameters **domain** (*str*) – domain name for the authorization

Returns a valid authzr

Return type acme.messages.AuthorizationResource

Raises

- ***acmems.exceptions.AuthorizationNotYetProcessed*** – We have to wait while the ACME server processes the autzr
- ***acmems.exceptions.AuthorizationNotYetRequested*** – new authzr created; have to wait until someone requests it
- ***acmems.exceptions.ChallengesUnknownStatus*** – unknown status
- ***acmems.exceptions.NoChallengeMethodsSupported*** – HTTP01 is not supported

1.2 Server

class acmems.server.**ACMEAbstractHandler**(*request*, *client_address*, *server*)
Bases: http.server.BaseHTTPRequestHandler

send_data(*data*, *content_type='text/plain'*, *response_code=200*)
Helper method to send data as HTTP response. The data are transfered as *text/plain*.

Parameters

- **data** (*str*) – The text to send as Python String.
- **response_code** (*int*) – HTTP response code

class acmems.server.**ACMEHTTPHandler**(*validator*, **args*, ***kwargs*)
Bases: *acmems.server.ACMEAbstractHandler*

do_GET()
Handles POST request (upload files).

class acmems.server.**ACMEMgmtHandler**(*request*, *client_address*, *server*)
Bases: *acmems.server.ACMEAbstractHandler*

do_POST()
Handles POST request (upload files).

1.3 Authentication & Processing

class acmems.auth.**SubjectAltName**(*componentType=None*, *tagSet=None*, *subtypeSpec=None*, *sizeSpec=None*)
Bases: ndg.httpsclient.subj_alt_name.SubjectAltName

ASN.1 implementation for subjectAltNames support

class acmems.auth.**IPAuthMethod**(*ips=None*)
Bases: object

Autentication by source IP

class acmems.auth.**HmacAuthMethod**
Bases: object

Authentication by HMAC / secret key

class acmems.auth.**AllAuthMethod**
Bases: object

Allow all authentication

class acmems.auth.**Block**(*name*, *options*, *config*)
Bases: object

One authentication block - combination of authentications and list of allowed domains

class acmems.auth.**Processor**(*auth*, *client_address*, *headers*, *rfile*)
Bases: object

Helper object to process a request, check authentication, reads and parse CSR

acceptable()
process the given request parameter for a CSR signing request and decide whether this request is allowed or not.

Parameters

- **str** (*client_ip*) – The source IP of the client (TCP level)
- **headers** (*dict*) – The request header
- **get_body** (*callable*) – function to read in body (CSR)

Return bool whether request should be accepted

1.4 Configuration

1.5 Exceptions

exception acmems.exceptions.**AcmeException**
Bases: Exception

Base exception call to be able to catch all ACMEMS specific errors

exception acmems.exceptions.**NoChallengeMethodsSupported**
Bases: *acmems.exceptions.AcmeException*

The domain can not be validated HTTP01

exception acmems.exceptions.**ChallengeFailed**(*domain*, *message*, *challenge_uri*)
Bases: *acmems.exceptions.AcmeException*

The challenge to validate the requested domain failed.

Variables

- **domain** (*str*) – the domain which the challenge should validate
- **message** (*str*) – message description from ACME server
- **challenge_uri** (*str*) – the URI of the failed challenge

exception acmems.exceptions.**ChallengesUnknownStatus**
Bases: *acmems.exceptions.AcmeException*

We do not known the status of the challenge. No clue what to do

exception acmems.exceptions.**AuthorizationNotYetProcessed**(*wait_until*)
Bases: *acmems.exceptions.AcmeException*

The authorization is be processed; until the next refresh it should at least be wait until `wait_until`

Variables **wait_until** (*datetime.datetime*) – first allowed retry time

exception acmems.exceptions.**AuthorizationNotYetRequested**(*event*)
Bases: *acmems.exceptions.AcmeException*

The newly created authorization challenge, was installed, but has not yet been requested by any client and is therefore currently pending or invalid.

Variables **event** (*threading.Event*) – event that will be signaled if someone requests the challenge.

exception acmems.exceptions.**RateLimited**
Bases: *acmems.exceptions.AcmeException*

To many requests

exception acmems.exceptions.**AccountError**
Bases: *acmems.exceptions.AcmeException*

Generic account error - e.g. - could not read private key - could not refresh the registration

exception acmems.exceptions.**NeedToAgreeToTOS**(*url*)
Bases: *acmems.exceptions.AccountError*

We are registered at the ACME server. But to use it, we need to accept the "Terms of Service"

exception acmems.exceptions.**InvalidDomainName**(*domain*, *detail*)
Bases: *acmems.exceptions.AcmeException*

The domain name is not excepted by the ACME server.

Variables

- **domain** (*str*) – the domain that was rejected
- **detail** (*str*) – the reject reason as string

exception acmems.exceptions.**PayloadToLarge**(*size*, *allowed*)
Bases: *acmems.exceptions.AcmeException*

The payload (CSR) it to large

Variables

- **size** (*int*) – the request size to upload (in bytes)
- **allowed** (*int*) – the maximal size in bytes

exception acmems.exceptions.**PayloadInvalid**
Bases: *acmems.exceptions.AcmeException*

The payload is not a valid CSR

CHAPTER 2

ChangeLog

This page lists all versions with its changes. ACMEMS follows Semantic Versioning.

2.1 Version 0

2.1.1 v0.3.1

Multiple bug fixes:

- Fix auth-block specific storage and verification settings
- IOError when replace certification in file storage
- Fix typos in dns01-dnsUpdate verification

2.1.2 v0.3.0

(Experimental) support for DNS challenges

2.1.3 v0.2.0

Reaching base architecture for 1.0 release. This includes:

- Restucture code and! *config* to support multiple verification mechanism
- WIP: experiment / prepare for dns01 challenge support (via dns updates)
- add storage support to not reissue CSRs the same pem, supporting reissue from multiple machines via a once shared key and CSR
- support newer python-acme releases

2.1.4 v0.1.1

- Fix syntax error in setup.py, preventing to upload to PyPI

2.1.5 v0.1.0

Implement basic feature set:

- submit CSR
- validate domain via HTTP
- sign certificate
- authenticate clients based on IP and HMAC

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

a