txacme Documentation

Release 0.9.0+0.g14e475b.dirty

Tristan Seligmann

August 13, 2016

Contents

ACME is Automatic Certificate Management Environment, a protocol that allows clients and certificate authorities to automate verification and certificate issuance. The ACME protocol is used by the free Let's Encrypt Certificate Authority.

`txacme` is an implementation of the protocol for Twisted, the event-driven networking engine for Python.

`txacme` is still under heavy development, and currently only an implementation of the client side of the protocol is planned; if you are interested in implementing or have need of the server side, please get in touch!

`txacme`'s documentation lives at Read the Docs, the code on GitHub. It's rigorously tested on Python 2.7, 3.4+, and PyPy.

CHAPTER 1

API entry points

There are several possible starting points for making use of txacme.

1.1 Server endpoint string

The simplest part of txacme to use is the stream server endpoint. Two endpoint parsers are provided, under the `le:` (Let's Encrypt) and `lets:` (Let's Encrypt Test in Staging) prefixes. The endpoint takes as parameters a directory to store certificates in, and the underlying endpoint to listen on.

Note: The Let's Encrypt staging environment generates certificates signed by *Fake LE Intermediate X1*, but does not have the stringent limits that the production environment has, so using it for testing before switching to the production environment is highly recommended.

A typical example:

```
twistd -n web --port lets:/srv/www/certs:tcp:443 --path /srv/www/root
```

Note: The certificate directory must already exist, and be writable by the user the application is running as.

The ACME client key will be stored in `client.key` in the cert directory. If this file does not exist, a new key will automatically be generated. Certificates (and chain certificates and keys) in PEM format will be stored in files named like `some.domain.name.pem` in the certificate directory. The appropriate certificate will be selected based on the servername that the client sends by SNI, so clients that do not perform SNI will not be able to connect.

In the event that there is no existing certificate available for a domain, an empty file can be used. This will be treated the same way as an expired certificate, and a new certificate will then be issued on startup. For example:

```
touch /srv/www/certs/example.com.pem
```

Note: This endpoint uses the `tls-sni-01` challenge type to perform authorization; this requires that the endpoint is reachable on port 443 for those domains (possibly via port forwarding).

Note: A certificate directory can be shared amongst multiple applications, using `le:` for the application running on port 443 to keep the certificates up to date, and `txsni:` for the other applications to make use of the same certificates.

At startup, and periodically (every 24 hours), a check will be performed for expiring certificates; if a certificate will expire in less than 30 days' time, it will be reissued. If the reissue fails, it will be retried at the next check. If a certificate will expire in less than 15 days' time, and reissue fails, a message will be logged at *CRITICAL* level.

1.2 Server endpoint API

The endpoint can be instantiated directly as well; this allows extra customizations beyond what the string syntax provides for. Most of the parameters that can be passed correspond to the parameters of the issuing service (see below).

class `txacme.endpoint.`**`AutoTLSEndpoint`**(*reactor*, *directory*, *client_creator*, *cert_store*, *cert_mapping*, *sub_endpoint*, *check_interval=datetime.timedelta(1)*, *reissue_interval=datetime.timedelta(30)*, *panic_interval=datetime.timedelta(15)*, *panic=<function _default_panic>*, *generate_key=<functools.partial object>*)

A server endpoint that does TLS SNI, with certificates automatically (re)issued from an ACME certificate authority.

Parameters

- **`reactor`** – The Twisted reactor.
- **`directory`** – `twisted.python.url.URL` for the ACME directory to use for issuing certs.
- **`client_creator`** (Callable[[reactor, `twisted.python.url.URL`], Deferred[*`txacme.client.Client`*]]) – A callable called with the reactor and directory URL for creating the ACME client. For example, `partial(Client.from_url, key=acme_key, alg=RS256)`.
- **`cert_store`** (`ICertificateStore`) – The certificate store containing the certificates to manage. For example, *`txacme.store.DirectoryStore`*.
- **`cert_mapping`** (*`dict`*) – The certificate mapping to use for SNI; for example, `txsni.snimap.HostDirectoryMap`. Usually this should correspond to the same underlying storage as `cert_store`.
- **`check_interval`** (*`timedelta`*) – How often to check for expiring certificates.
- **`reissue_interval`** (*`timedelta`*) – If a certificate is expiring in less time than this interval, it will be reissued.
- **`panic_interval`** (*`timedelta`*) – If a certificate is expiring in less time than this interval, and reissuing fails, the panic callback will be invoked.
- **`panic`** (Callable[[Failure, `str`], Deferred]) – A callable invoked with the failure and server name when reissuing fails for a certificate expiring in the `panic_interval`. For example, you could generate a monitoring alert. The default callback logs a message at *CRITICAL* level.
- **`generate_key`** – A 0-arg callable used to generate a private key for a new cert. Normally you would not pass this unless you have specialized key generation requirements.

`listen`(*protocolFactory*)

Start an issuing service, and wait until initial issuing is complete.

1.3 Issuing service

The endpoint is a simple wrapper that combines the functionality of the txsni endpoint for handling SNI, and the issuing service which takes care of (re)issuing certificates using an ACME service.

class txacme.service.**AcmeIssuingService**(*cert_store*, *client*, *clock*, *responders*, *check_interval=datetime.timedelta(1)*, *reissue_interval=datetime.timedelta(30)*, *panic_interval=datetime.timedelta(15)*, *panic=<function _default_panic>*, *generate_key=<functools.partial object>*, *waiting=NOTHING*)

A service for keeping certificates up to date by using an ACME server.

Parameters

- **cert_store** (ICertificateStore) – The certificate store containing the certificates to manage.
- **client** (Client) – The ACME client to use. Typically constructed with *Client.from_url*.
- **clock** – IReactorTime provider; usually the reactor, when not testing.
- **responders** (List[*IResponder*]) – Challenge responders. Usually only one responder is needed; if more than one responder for the same type is provided, only the first will be used.
- **check_interval** (*timedelta*) – How often to check for expiring certificates.
- **reissue_interval** (*timedelta*) – If a certificate is expiring in less time than this interval, it will be reissued.
- **panic_interval** (*timedelta*) – If a certificate is expiring in less time than this interval, and reissuing fails, the panic callback will be invoked.
- **panic** (Callable[[Failure, str], Deferred]) – A callable invoked with the failure and server name when reissuing fails for a certificate expiring in the panic_interval. For example, you could generate a monitoring alert. The default callback logs a message at *CRITICAL* level.
- **generate_key** – A 0-arg callable used to generate a private key for a new cert. Normally you would not pass this unless you have specialized key generation requirements.

when_certs_valid()

Get a notification once the startup check has completed.

When the service starts, an initial check is made immediately; the deferred returned by this function will only fire once reissue has been attempted for any certificates within the panic interval.

Note: The reissue for any of these certificates may not have been successful; the panic callback will be invoked for any certificates in the panic interval that failed reissue.

Return type Deferred

Returns A deferred that fires once the initial check has resolved.

The *ICertificateStore* and *IResponder* interfaces are the main extension points for using the issuing service directly. For example, a custom implementation might manage the certificate configuration of a cloud load balancer, implementing the `dns-01` challenge type by modifying DNS entries in the cloud DNS configuration.

CHAPTER 2

Other documentation

2.1 API stability

txacme is versioned according to SemVer 2.0.0. In addition, since SemVer does not make this explicit, versions following txacme 1.0.0 will have a "rolling compatibility" guarantee: new major versions will not break behaviour that did not already emit a deprecation warning in the latest minor version of the previous major version series.

The current version number of 0.9.x is intended to reflect the not-quite-finalized nature of the API. While it is not expected that the API will change drastically, the 0.9 version series is intended to allow space for users to experiment and identify any issues obstructing their use cases so that these can be corrected before the API is finalized in the 1.0.0 release.

2.2 txacme changelog

2.2.1 Txacme 0.9.0 (2016-04-10)

Features

- Initial release! (#23)

2.3 txacme

2.3.1 txacme package

Subpackages

txacme.test package

Submodules

txacme.test.matchers module

class `txacme.test.matchers.`**`ValidForName`**(*name*)

Matches when the matchee object (must be a `Certificate` or `CertificateSigningRequest`) is valid for the given name.

txacme.test.strategies module Miscellaneous strategies for Hypothesis testing.

`txacme.test.strategies.dns_labels()`
Strategy for generating limited charset DNS labels.

`txacme.test.strategies.dns_names()`
Strategy for generating limited charset DNS names.

`txacme.test.strategies.urls()`
Strategy for generating `twisted.python.url.URL`s.

txacme.test.test_challenges module Tests for *`txacme.challenges`*.

class `txacme.test.test_challenges.ResponderTests(*args, **kwargs)`
`TLSSNI01Responder` is a responder for tls-sni-01 challenges that works with txsni.

`test_interface()`
The *`IResponder`* interface is correctly implemented.

`test_start_responding()`
Calling `start_responding` makes an appropriate entry appear in the host map.

`test_stop_responding_already_stopped()`
Calling `stop_responding` when we are not responding for a server name does nothing.

txacme.test.test_client module

class `txacme.test.test_client.ClientTests(*args, **kwargs)`
`Client` provides a client interface for the ACME API.

`test_agree_to_tos()`
Agreeing to the TOS returns a registration with the agreement updated.

`test_answer_challenge()`
`answer_challenge` responds to a challenge and returns the updated challenge.

`test_answer_challenge_function()`
The challenge is found in the responder after invoking *`answer_challenge`*.

`test_authorization_missing_link()`
`_parse_authorization` raises `ClientError` if the `"next"` link is missing.

`test_authorization_unexpected_identifier()`
`_check_authorization` raises `UnexpectedUpdate` if the return identifier doesn't match.

`test_challenge_missing_link()`
`_parse_challenge` raises `ClientError` if the `"up"` link is missing.

`test_challenge_unexpected_uri()`
`_check_challenge` raises `UnexpectedUpdate` if the challenge does not have the expected URI.

`test_default_client()`
`~txacme.client._default_client` constructs a client if one was not provided.

`test_expect_response_wrong_code()`
`_expect_response` raises `ClientError` if the response code does not match the expected code.

`test_fetch_chain_empty()`
If a certificate has no issuer link, *`Client.fetch_chain`* returns an empty chain.

`test_fetch_chain_okay()`
A certificate chain that is shorter than the max length is returned.

test_fetch_chain_too_long()
A certificate chain that is too long fails with `ClientError`.

test_fqdn_identifier()
fqdn_identifier constructs an `Identifier` of the right type.

test_from_directory()
from_url() constructs a client with a directory retrieved from the given URL.

test_no_tls_sni_01()
If no tls-sni-01 challenges are available, *NoSupportedChallenges* is raised.

test_only_tls_sni_01()
If a singleton tls-sni-01 challenge is available, it is returned.

test_poll()
poll retrieves the latest state of an authorization resource, as well as the minimum time to wait before polling the state again.

test_poll_invalid()
If the authorization enters an invalid state while polling, *poll_until_valid* will fail with *AuthorizationFailed*.

test_poll_timeout()
If the timeout is exceeded during polling, *poll_until_valid* will fail with `CancelledError`.

test_poll_valid()
If the authorization enters a valid state while polling, *poll_until_valid* will fire with the updated authorization.

test_register()
If the registration succeeds, the new registration is returned.

test_register_error()
If some other error occurs during registration, a *txacme.client.ServerError* results.

test_register_existing()
If registration fails due to our key already being registered, the existing registration is returned.

test_register_missing_next()
If the directory does not return a `"next"` link, a `ClientError` failure occurs.

test_request_challenges()
request_challenges() creates a new authorization, and returns the authorization resource with a list of possible challenges to proceed with.

test_request_issuance()
If issuing is successful, a certificate resource is returned.

test_tls_sni_01_no_singleton()
If a suitable singleton challenge is not found, *NoSupportedChallenges* is raised.

test_unexpected_update()
If the server does not return the registration we expected, an `UnexpectedUpdate` failure occurs.

class txacme.test.test_client.**ExtraCoverageTests**(**args*, ***kwargs*)
Tests to get coverage on some test helpers that we don't really want to maintain ourselves.

test_consume_context_manager_fails_on_remaining_requests()
If the `consume` context manager is used, if there are any remaining expecting requests, the test case will be failed.

test_unexpected_number_of_request_causes_failure()
If there are no more expected requests, making a request causes a failure.

class `txacme.test.test_client.`**`LinkParsingTests`**(**args*, ***kwargs*)

`_parse_header_links` parses the links from a response with Link: header fields. This implementation is ... actually not very good, which is why there aren't many tests.

`test_rfc_example1`()

The first example from the RFC.

txacme.test.test_endpoint module Tests for *`txacme.endpoint`*.

class `txacme.test.test_endpoint.`**`EndpointTests`**(**args*, ***kwargs*)

Tests for *`AutoTLSEndpoint`*.

`test_listen_starts_service`()

`AutoTLSEndpoint.listen` starts an `AcmeIssuingService`. Stopping the port stops the service.

class `txacme.test.test_endpoint.`**`PluginTests`**(**args*, ***kwargs*)

Tests for the plugins.

`test_le_parser`()

The `le:` parser uses the Let's Encrypt production directory, and provides the relevant interfaces.

`test_lets_parser`()

The `lets:` parser uses the Let's Encrypt staging directory, and provides the relevant interfaces.

`test_parser`()

`AcmeParser` creates an endpoint with the specified ACME directory and directory store.

txacme.test.test_matchers module

class `txacme.test.test_matchers.`**`ValidForNameTests`**(**args*, ***kwargs*)

`ValidForName` matches if a CSR/cert is valid for the given name.

txacme.test.test_service module

class `txacme.test.test_service.`**`AcmeIssuingServiceTests`**(**args*, ***kwargs*)

Tests for *`txacme.service.AcmeIssuingService`*.

`test_blank_cert`()

An empty certificate file will be treated like an expired certificate.

`test_default_panic`()

The default panic callback logs a message via `twisted.logger`.

`test_errors`()

If a cert renewal fails within the panic interval, the panic callback is invoked; otherwise the error is logged normally.

`test_starting_stopping_cancellation`()

Test the starting and stopping behaviour.

`test_time_marches_on`()

Any certs that have exceeded the panic or reissue intervals will be reissued at the next check.

`test_when_certs_valid_all_certs_valid`()

The deferred returned by `when_certs_valid` fires immediately if none of the certs in the store are expired.

`test_when_certs_valid_certs_expired`()

The deferred returned by `when_certs_valid` only fires once all panicing and expired certs have been renewed.

test_when_certs_valid_no_certs()
The deferred returned by `when_certs_valid` fires immediately if there are no certs in the store.

txacme.test.test_store module

class txacme.test.test_store.**DirectoryStoreTests**(**args*, ***kwargs*)
Tests for *txacme.store.DirectoryStore*.

class txacme.test.test_store.**MemoryStoreTests**(**args*, ***kwargs*)
Tests for *txacme.testing.MemoryStore*.

txacme.test.test_util module

class txacme.test.test_util.**GeneratePrivateKeyTests**(**args*, ***kwargs*)
generate_private_key generates private keys of various types using sensible parameters.

test_rsa_key()
Passing `u'rsa'` results in an RSA private key.

test_unknown_key_type()
Passing an unknown key type results in `ValueError`.

class txacme.test.test_util.**GenerateCertTests**(**args*, ***kwargs*)
generate_tls_sni_01_cert generates a cert and key suitable for responding for the given challenge SAN.

test_cert_verifies()
The certificates generated verify using `verify_cert`.

class txacme.test.test_util.**CSRTests**(**args*, ***kwargs*)
encode_csr and *decode_csr* serialize CSRs in JOSE Base64 DER encoding.

test_common_name_too_long()
If the first name provided is too long, *csr_for_names* uses a dummy value for the common name.

test_decode_garbage()
If decoding fails, *decode_csr* raises `DeserializationError`.

test_empty_names_invalid()
csr_for_names raises `ValueError` if given an empty list of names.

test_roundtrip()
The encoding roundtrips.

test_valid_for_names()
csr_for_names returns a CSR that is actually valid for the given names.

Module contents

Submodules

txacme.challenges module

Implementations of ACME challenge mechanisms.

See also:

`acme.challenges`

class txacme.challenges.**TLSSNI01Responder**
A tls-sni-01 challenge responder for txsni.

start_responding(*response*)
Put a context into the mapping.

stop_responding(*response*)
Remove a context from the mapping.

wrap_host_map(*host_map*)
Wrap a txsni host mapping.

The wrapper should be passed to `txsni.snimap.SNIMap`; any active challenge server names will override entries in the wrapped map, but this scenario is unlikely to occur due to the invalid nature of these names.

txacme.client module

ACME client API (like `acme.client`) implementation for Twisted.

class `txacme.client.`**Client**(*directory*, *reactor*, *key*, *jws_client*)
ACME client interface.

agree_to_tos(*regr*)
Accept the terms-of-service for a registration.

Parameters **regr** (*RegistrationResource*) – The registration to update.

Returns The updated registration resource.

Return type Deferred[`RegistrationResource`]

answer_challenge(*challenge_body*, *response*)
Respond to an authorization challenge.

Parameters

- **challenge_body** (*ChallengeBody*) – The challenge being responded to.
- **response** (*ChallengeResponse*) – The response to the challenge.

Returns The updated challenge resource.

Return type Deferred[`ChallengeResource`]

fetch_chain(*certr*, *max_length=10*)
Fetch the intermediary chain for a certificate.

Parameters

- **certr** (*acme.messages.CertificateResource*) – The certificate to fetch the chain for.
- **max_length** (*int*) – The maximum length of the chain that will be fetched.

Return type Deferred[List[`acme.messages.CertificateResource`]]

Returns The issuer certificate chain, ordered with the trust anchor last.

classmethod from_url(*reactor*, *url*, *key*, *alg=RS256*, *jws_client=None*)
Construct a client from an ACME directory at a given URL.

Parameters

- **url** – The `twisted.python.url.URL` to fetch the directory from.
- **reactor** – The Twisted reactor to use.

- **key** (*JWK*) – The client key to use.
- **alg** – The signing algorithm to use. Needs to be compatible with the type of key used.
- **jws_client** (JWSClient) – The underlying client to use, or None to construct one.

Returns The constructed client.

Return type Deferred[*Client*]

poll(*authzr*)
Update an authorization from the server (usually to check its status).

register(*new_reg=None*)
Create a new registration with the ACME server.

Parameters **new_reg** (*NewRegistration*) – The registration message to use, or None to construct one.

Returns The registration resource.

Return type Deferred[RegistrationResource]

request_challenges(*identifier*)
Create a new authorization.

Parameters **identifier** (*Identifier*) – The identifier to authorize.

Returns The new authorization resource.

Return type Deferred[AuthorizationResource]

request_issuance(*csr*)
Request a certificate.

Authorizations should have already been completed for all of the names requested in the CSR.

Note that unlike acme.client.Client.request_issuance, the certificate resource will have the body data as raw bytes.

See also:

txacme.util.csr_for_names

Todo

Delayed issuance is not currently supported, the server must issue the requested certificate immediately.

Parameters **csr** – A certificate request message: normally *txacme.messages.CertificateRequest* or acme.messages.CertificateRequest.

Return type Deferred[acme.messages.CertificateResource]

Returns The issued certificate.

classmethod retry_after(*response*, *default=5*, *_now=<built-in function time>*)
Parse the Retry-After value from a response.

update_registration(*regr*, *uri=None*)
Submit a registration to the server to update it.

Parameters

- **regr** (*RegistrationResource*) – The registration to update. Can be a NewRegistration instead, in order to create a new registration.
- **uri** (*str*) – The url to submit to. Must be specified if a NewRegistration is provided.

Returns The updated registration resource.

Return type Deferred[RegistrationResource]

class txacme.client.**JWSClient**(*treq_client*, *key*, *alg*, *user_agent='txacme/0.9.0+0.g14e475b.dirty'*)

HTTP client using JWS-signed messages.

get(*url*, *content_type='application/json'*, ***kwargs*)

Send GET request and check response.

Parameters

- **method** (*str*) – The HTTP method to use.
- **url** (*str*) – The URL to make the request to.

Raises

- ***txacme.client.ServerError*** – If server response body carries HTTP Problem (draft-ietf-appsawg-http-problem-00).
- **acme.errors.ClientError** – In case of other protocol errors.

Returns Deferred firing with the checked HTTP response.

head(*url*, **args*, ***kwargs*)

Send HEAD request without checking the response.

Note that _check_response is not called, as there will be no response body to check.

Parameters **url** (*str*) – The URL to make the request to.

post(*url*, *obj*, *content_type='application/json'*, ***kwargs*)

POST an object and check the response.

Parameters

- **url** (*str*) – The URL to request.
- **obj** (*JSONDeSerializable*) – The serializable payload of the request.
- **content_type** (*bytes*) – The expected content type of the response. By default, JSON.

Raises

- ***txacme.client.ServerError*** – If server response body carries HTTP Problem (draft-ietf-appsawg-http-problem-00).
- **acme.errors.ClientError** – In case of other protocol errors.

exception txacme.client.**ServerError**(*message*, *response*)

acme.messages.Error isn't usable as an asynchronous exception, because it doesn't allow setting the __traceback__ attribute like Twisted wants to do when cleaning Failures. This type exists to wrap such an error, as well as provide access to the original response.

txacme.client.**fqdn_identifier**(*fqdn*)

Construct an identifier from an FQDN.

Trivial implementation, just saves on typing.

Parameters **fqdn** (*str*) – The domain name.

Returns The identifier.

Return type Identifier

`txacme.client.`**`answer_challenge`**(*authzr*, *client*, *responders*)

Complete an authorization using a responder.

Parameters

- **auth** (*AuthorizationResource*) – The authorization to complete.
- **client** (Client) – The ACME client.
- **responders** (List[*IResponder*]) – A list of responders that can be used to complete the challenge with.

Returns A deferred firing when the authorization is verified.

`txacme.client.`**`poll_until_valid`**(*authzr*, *clock*, *client*, *timeout=300.0*)

Poll an authorization until it is in a state other than pending or processing.

Parameters

- **auth** (*AuthorizationResource*) – The authorization to complete.
- **clock** – The IReactorTime implementation to use; usually the reactor, when not testing.
- **client** (Client) – The ACME client.
- **timeout** (*float*) – Maximum time to poll in seconds, before giving up.

Raises ***txacme.client.AuthorizationFailed*** – if the authorization is no longer in the pending, processing, or valid states.

Raises twisted.internet.defer.CancelledError if the authorization was still in pending or processing state when the timeout was reached.

Return type Deferred[AuthorizationResource]

Returns A deferred firing when the authorization has completed/failed; if the authorization is valid, the authorization resource will be returned.

exception `txacme.client.`**`NoSupportedChallenges`**

No supported challenges were found in an authorization.

exception `txacme.client.`**`AuthorizationFailed`**(*authzr*)

An attempt was made to complete an authorization, but it failed.

txacme.endpoint module

A TLS endpoint that supports SNI automatically issues / renews certificates via an ACME CA (eg. Let's Encrypt).

class `txacme.endpoint.`**`AutoTLSEndpoint`**(*reactor*, *directory*, *client_creator*, *cert_store*, *cert_mapping*, *sub_endpoint*, *check_interval=datetime.timedelta(1)*, *reissue_interval=datetime.timedelta(30)*, *panic_interval=datetime.timedelta(15)*, *panic=<function _default_panic>*, *generate_key=<functools.partial object>*)

A server endpoint that does TLS SNI, with certificates automatically (re)issued from an ACME certificate authority.

Parameters

- **reactor** – The Twisted reactor.
- **directory** – `twisted.python.url.URL` for the ACME directory to use for issuing certs.
- **client_creator** (Callable[[reactor, `twisted.python.url.URL`], Deferred[*`txacme.client.Client`*]]) – A callable called with the reactor and directory URL for creating the ACME client. For example, `partial(Client.from_url, key=acme_key, alg=RS256)`.
- **cert_store** (`ICertificateStore`) – The certificate store containing the certificates to manage. For example, *`txacme.store.DirectoryStore`*.
- **cert_mapping** (*`dict`*) – The certificate mapping to use for SNI; for example, `txsni.snimap.HostDirectoryMap`. Usually this should correspond to the same underlying storage as `cert_store`.
- **check_interval** (*`timedelta`*) – How often to check for expiring certificates.
- **reissue_interval** (*`timedelta`*) – If a certificate is expiring in less time than this interval, it will be reissued.
- **panic_interval** (*`timedelta`*) – If a certificate is expiring in less time than this interval, and reissuing fails, the panic callback will be invoked.
- **panic** (Callable[[Failure, `str`], Deferred]) – A callable invoked with the failure and server name when reissuing fails for a certificate expiring in the `panic_interval`. For example, you could generate a monitoring alert. The default callback logs a message at *CRITICAL* level.
- **generate_key** – A 0-arg callable used to generate a private key for a new cert. Normally you would not pass this unless you have specialized key generation requirements.

listen(*protocolFactory*)

Start an issuing service, and wait until initial issuing is complete.

txacme.interfaces module

Interface definitions for txacme.

interface `txacme.interfaces.`**IResponder**

Configuration for a ACME challenge responder.

The actual responder may exist somewhere else, this interface is merely for an object that knows how to configure it.

challenge_type

The type of challenge this responder is able to respond for.

Must correspond to one of the types from `acme.challenges`; for example, `u'tls-sni-01'`.

stop_responding(*server_name*)

Stop responding for a particular challenge.

May be a noop if a particular responder does not need or implement explicit cleanup; implementations should not rely on this method always being called.

Parameters **response** – The `acme.challenges` response object; the exact type of this object depends on the challenge type.

start_responding(*response*)
Start responding for a particular challenge.

Parameters **response** – The `acme.challenges` response object; the exact type of this object depends on the challenge type.

Return type `Deferred`

Returns A deferred firing when the challenge is ready to be verified.

interface `txacme.interfaces.`**ICertificateStore**
A store of certificate/keys/chains.

as_dict(*self*)
Get all certificates in the store.

Return type `Deferred[Dict[str, List[:ref:`pem-objects`]]]`

Returns A deferred firing with a dict mapping server names to PEM Objects.

store(*self*, *server_name*, *pem_objects*)
Store PEM objects for the given server name.

Implementations do not have to permit invoking this with a server name that was not already present in the store.

Parameters

- **server_name** (*str*) – The server name to update.
- **pem_objects** – A list of PEM Objects; must contain exactly one private key, a certificate corresponding to that private key, and zero or more chain certificates.

Return type `Deferred`

get(*self*, *server_name*)
Retrieve the current PEM objects for the given server name.

Parameters **server_name** (*str*) – The server name.

Raises **KeyError** – if the given name does not exist in the store.

Returns `Deferred[List[:ref:`pem-objects`]]`

txacme.logging module

Eliot message and action definitions.

txacme.messages module

ACME protocol messages.

This module provides supplementary message implementations that are not already provided by the `acme` library.

See also:

`acme.messages`

class `txacme.messages.`**CertificateRequest**(***kwargs*)
ACME new-cert request.

Differs from the upstream version because it wraps a Cryptography CSR object instead of a PyOpenSSL one.

See also:

`acme.messages.CertificateRequest`, `cryptography.x509.CertificateSigningRequest`

txacme.service module

class `txacme.service.`**`AcmeIssuingService`**(*cert_store*, *client*, *clock*, *responders*, *check_interval=datetime.timedelta(1)*, *reissue_interval=datetime.timedelta(30)*, *panic_interval=datetime.timedelta(15)*, *panic=<function _default_panic>*, *generate_key=<functools.partial object>*, *waiting=NOTHING*)

A service for keeping certificates up to date by using an ACME server.

Parameters

- **`cert_store`** (`ICertificateStore`) – The certificate store containing the certificates to manage.
- **`client`** (`Client`) – The ACME client to use. Typically constructed with `Client.from_url`.
- **`clock`** – `IReactorTime` provider; usually the reactor, when not testing.
- **`responders`** (List[`IResponder`]) – Challenge responders. Usually only one responder is needed; if more than one responder for the same type is provided, only the first will be used.
- **`check_interval`** (`timedelta`) – How often to check for expiring certificates.
- **`reissue_interval`** (`timedelta`) – If a certificate is expiring in less time than this interval, it will be reissued.
- **`panic_interval`** (`timedelta`) – If a certificate is expiring in less time than this interval, and reissuing fails, the panic callback will be invoked.
- **`panic`** (Callable[[Failure, `str`], Deferred]) – A callable invoked with the failure and server name when reissuing fails for a certificate expiring in the `panic_interval`. For example, you could generate a monitoring alert. The default callback logs a message at *CRITICAL* level.
- **`generate_key`** – A 0-arg callable used to generate a private key for a new cert. Normally you would not pass this unless you have specialized key generation requirements.

`when_certs_valid`()

Get a notification once the startup check has completed.

When the service starts, an initial check is made immediately; the deferred returned by this function will only fire once reissue has been attempted for any certificates within the panic interval.

Note: The reissue for any of these certificates may not have been successful; the panic callback will be invoked for any certificates in the panic interval that failed reissue.

Return type `Deferred`

Returns A deferred that fires once the initial check has resolved.

txacme.store module

`txacme.interfaces.ICertificateStore` implementations.

class `txacme.store.`**`DirectoryStore`**(*path*)
A certificate store that keeps certificates in a directory on disk.

txacme.testing module

Utilities for testing with txacme.

class `txacme.testing.`**`FakeClient`**(*key*, *clock*, *ca_key=None*)
Provides the same API as `Client`, but performs no network operations and issues certificates signed by its own fake CA.

class `txacme.testing.`**`MemoryStore`**(*certs=None*)
A certificate store that keeps certificates in memory only.

class `txacme.testing.`**`NullResponder`**(*challenge_type*)
A responder that does absolutely nothing.

txacme.util module

Utility functions that may prove useful when writing an ACME client.

`txacme.util.`**`generate_private_key`**(*key_type*)
Generate a random private key using sensible parameters.

Parameters **`key_type`** (`str`) – The type of key to generate. One of: `rsa`.

`txacme.util.`**`generate_tls_sni_01_cert`**(*server_name*, *key_type=u'rsa'*, *_generate_private_key=None*)
Generate a certificate/key pair for responding to a tls-sni-01 challenge.

Parameters

- **`server_name`** (`str`) – The SAN the certificate should have.
- **`key_type`** (`str`) – The type of key to generate; usually not necessary.

Return type `Tuple[`~cryptography.x509.Certificate`, PrivateKey]`

Returns A tuple of the certificate and private key.

`txacme.util.`**`cert_cryptography_to_pyopenssl`**(*cert*)
Convert a `cryptography.x509.Certificate` object to an `OpenSSL.crypto.X509` object.

`txacme.util.`**`key_cryptography_to_pyopenssl`**(*key*)
Convert a Cryptography private key object to an `OpenSSL.crypto.PKey` object.

`txacme.util.`**`tap`**(*f*)
"Tap" a Deferred callback chain with a function whose return value is ignored.

`txacme.util.`**`encode_csr`**(*csr*)
Encode CSR as JOSE Base-64 DER.

Parameters **`csr`** (`cryptography.x509.CertificateSigningRequest`) – The CSR.

Return type str

`txacme.util.`**`decode_csr`**(*b64der*)

Decode JOSE Base-64 DER-encoded CSR.

Parameters **`b64der`** (*`str`*) – The encoded CSR.

Return type `cryptography.x509.CertificateSigningRequest`

Returns The decoded CSR.

`txacme.util.`**`csr_for_names`**(*names*, *key*)

Generate a certificate signing request for the given names and private key.

See also:

`acme.client.Client.request_issuance`

See also:

`generate_private_key`

Parameters

- **`List[str]`** – One or more names (subjectAltName) for which to request a certificate.
- **`key`** – A Cryptography private key object.

Return type `cryptography.x509.CertificateSigningRequest`

Returns The certificate request message.

`txacme.util.`**`clock_now`**(*clock*)

Get a datetime representing the current time.

Parameters **`clock`** – An `IReactorTime` provider.

Return type `datetime`

Returns A datetime representing the current time.

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

U

V

W