ska Documentation

Release 1.10

Artur Barseghyan <artur.barseghyan@gmail.com>

Aug 27, 2023

CONTENTS

Lets you easily sign data, using symmetric-key algorithm encryption. Allows you to validate signed data and identify possible validation errors. Uses sha-(1, 224, 256, 385 and 512)/hmac for signature encryption. Allows to use custom hash algorithms. Comes with shortcut functions for signing (and validating) dictionaries and URLs.

CHAPTER ONE

KEY CONCEPTS

Hosts, that communicate with each other, share the Secret Key, which is used to sign data (requests). Secret key is never sent around.

One of the cases is signing of HTTP requests. Each (HTTP) request is signed on the sender side using the shared Secret Key and as an outcome produces the triple (`signature`, `auth_user`, `valid_until`) which are used to sign the requests.

- `signature` (`str`): Signature generated.
- `auth_user` (`str`): User making the request. Can be anything.
- `valid_until` (`float` or `str`): Signature expiration time (Unix timestamp).

On the recipient side, (HTTP request) data is validated using the shared Secret Key. It's being checked whether signature is valid and not expired.

```
┌─────────────┐           Data           ┌─────────────┐
│   Host 1    ├─────────────────────────>│   Host 2    │
│ ─────────── │                          │ ─────────── │
│ secret key  │                          │ secret key  │
│ 'my-secret' │<─────────────────────────┤ 'my-secret' │
└─────────────┘           Data           └─────────────┘
```

CHAPTER TWO

FEATURES

2.1 Core *ska* module

- Sign dictionaries.
- Validate signed dictionaries.
- Sign URLs. Append and sign additional URL data.
- Validate URLs.
- Use one of the built-in algorythms (HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384 or HMAC SHA-512) or define a custom one.

2.2 Django *ska* module (*ska.contrib.django.ska*)

- Model decorators for signing absolute URLs. View (including class-based views) decorators for protecting views to authorised parties only (no authentication required).
- Authentication backend for Django based on the signatures (tokens) generated using *ska*, which allows you to get a password-less login to Django web site. Multiple Secret Keys (per provider) supported. Comes with handy callbacks (possible to customise per provider) for various states of authentication.
- Template tags for signing URLs from within templates.
- *django-constance* integration (for password-less authentication).
- *Django REST Framework integration* (for protecting ViewSets, obtaining JWT tokens for authentication).

CHAPTER THREE

PREREQUISITES

3.1 Present

- Core `ska` module requires Python 3.8, 3.9, 3.10 and 3.11.
- Django `ska` module (`ska.contrib.django.ska`) requires the mentioned above plus Django 3.2, 4.1 or 4.2. Additionally, certain versions of *django-constance* and *djangorestframework* are required. Specific version requirement primarily depends on the used Django version. Check the example requirements to find out which versions of *django-constance* and *djangorestframework* have been tested with specific Django versions.

3.2 Past

Note: In future releases (any time) compatibility with no-longer-supported versions might/will be wiped out.

- Dropping support of Python 3.6 and 3.7 has been announced in version 1.10. As of 1.9.1 everything still worked.
- Dropping support of Python 2.7 and 3.5 has been announced in version 1.8. As of 1.7.5 everything still worked.
- Dropping support of Python 3.4 has been announced in version 1.6.8. As of 1.6.8 everything still worked.
- Dropping support of Django 2.2, 3.0, 3.1 and 4.0 has been announced in version 1.10. As of 1.9.1 everything is still backwards compatible with mentioned versions.
- Dropping support of Django 1.5, 1.6 and 1.7 has been announced in version 1.6. As of 1.6 everything is still backwards compatible with mentioned versions.
- Dropping support of Python 2.6 and 3.3 has been announced in version 1.6. As of 1.6 everything is still backwards compatible (as much as it's possible within this package) with mentioned versions.

CHAPTER FOUR

ECO-SYSTEM

Need `ska` for other languages? Check the following affiliated projects:

- skajs: `ska` implementation for NodeJS (both CommonJS and ESM are supported, Node >= 14).
- skaphp: `ska` implementation for PHP (>= 7.2).

Generated signatures are inter-compatible between Python, NodeJS and PHP implementations.

CHAPTER FIVE

INSTALLATION

Latest stable version from PyPI:

```
pip install ska
```

or latest development version from GitHub.

```
pip install https://github.com/barseghyanartur/ska/archive/main.tar.gz
```

CHAPTER SIX

USAGE EXAMPLES

For integration with Django, see the *Django integration* section.

6.1 Basic usage

Pure Python usage.

6.1.1 Sender side

Signing URLs is as simple as follows.

Required imports.

```
from ska import sign_url
```

Producing a signed URL.

```
signed_url = sign_url(
    auth_user='user',
    secret_key='your-secret_key',
    url='http://e.com/api/'
)
```

```
GET http://e.com/api/?valid_until=1378045287.0&auth_user=user&
↪signature=YlZpLFsjUKBalL4x5trhkeEgqE8%3D
```

Default lifetime of a signature is 10 minutes (600 seconds). If you want it to be different, provide a `lifetime` argument to `sign_url` function.

Default name of the (GET) param holding the generated signature value is `signature`. If you want it to be different, provide a `signature_param` argument to `sign_url` function.

Default name of the (GET) param holding the `auth_user` value is `auth_user`. If you want it to be different, provide a `auth_user_param` argument to `sign_url` function.

Default name of the (GET) param holding the `valid_until` value is *valid_until*. If you want it to be different, provide a `valid_until_param` argument to `sign_url` function.

Note, that by default a suffix '?' is added after the given `url` and generated signature params. If you want that suffix to be custom, provide a `suffix` argument to the `sign_url` function. If you want it to be gone, set its' value to empty string.

With all customisations, it would look as follows:

```
from ska import HMACSHA512Signature  # Use HMAC SHA-512 algorithm

signed_url = sign_url(
    auth_user='user',
    secret_key='your-secret_key',
    lifetime=120,
    url='http://e.com/api/',
    signature_param='signature',
    auth_user_param='auth_user',
    valid_until_param='valid_until',
    signature_cls=HMACSHA512Signature
)
```

It's also possible to add additional data to the signature by providing a `extra` argument (dict). Note, that additional data is signed as well. If request is somehow tampered (values vary from originally provided ones), signature becomes invalid.

```
sign_url(
    auth_user='user',
    secret_key='your-secret_key',
    url='http://e.com/api/',
    extra={
        'email': 'doe@example.com',
        'last_name': 'Doe',
        'first_name': 'Joe'
    }
)
```

You may now proceed with the signed URL request. If you use the famous `requests` library, it would be as follows.

```
import requests
requests.get(signed_url)
```

If you want to use POST method instead, you would likely want to get a dictionary back, in order to append it to the POST data later.

Required imports.

```
from ska import signature_to_dict
```

Producing a dictionary containing the signature data, ready to be put into the request (for example POST) data. All customisations mentioned above for the `sign_url` function, also apply to the `signature_to_dict`:

```
signature_dict = signature_to_dict(
    auth_user='user',
    secret_key='your-secret_key'
)
```

```
{
    'signature': 'YlZpLFsjUKBalL4x5trhkeEgqE8=',
    'auth_user': 'user',
    'valid_until': '1378045287.0'
}
```

Adding of additional data to the signature works in the same way:

```
signature_dict = signature_to_dict(
    auth_user='user',
    secret_key='your-secret_key',
    extra={
        'email': 'john.doe@mail.example.com',
        'first_name': 'John',
        'last_name': 'Doe'
    }
)
```

```
{
    'auth_user': 'user',
    'email': 'john.doe@mail.example.com',
    'extra': 'email,first_name,last_name',
    'first_name': 'John',
    'last_name': 'Doe',
    'signature': 'cnSoU/LnJ/ZhfLtDLzab3a3gkug=',
    'valid_until': 1387616469.0
}
```

If you for some reason prefer a lower level implementation, read the same section in the *Advanced usage (low-level)* chapter.

6.1.2 Recipient side

Validating the signed request data is as simple as follows.

Required imports.

```
from ska import validate_signed_request_data
```

Validating the signed request data. Note, that `data` value is expected to be a dictionary; `request.GET` is given as an example. It will most likely vary from what's used in your framework (unless you use Django).

```
validation_result = validate_signed_request_data(
    data=request.GET,  # Note, that ``request.GET`` is given as example.
    secret_key='your-secret_key'
)
```

The `validate_signed_request_data` produces a `ska.SignatureValidationResult` object, which holds the following data.

- `result` (`bool`): True if data is valid. False otherwise.
- `reason` (`list`): List of strings, indicating validation errors. Empty list in case if `result` is True.

Default name of the (GET) param holding the signature value is `signature`. If you want it to be different, provide a `signature_param` argument to `validate_signed_request_data` function.

Default name of the (GET) param holding the `auth_user` value is `auth_user`. If you want it to be different, provide a `auth_user_param` argument to `validate_signed_request_data` function.

Default name of the (GET) param holding the `valid_until` value is `valid_until`. If you want it to be different, provide a `valid_until_param` argument to `validate_signed_request_data` function.

With all customisations, it would look as follows. Note, that `request.GET` is given as example.

```
from ska import HMACSHA256Signature  # Use HMAC SHA-256 algorithm

validation_result = validate_signed_request_data(
    data=request.GET,
    secret_key='your-secret_key',
    signature_param='signature',
    auth_user_param='auth_user',
    valid_until_param='valid_until',
    signature_cls=HMACSHA256Signature
)
```

If you for some reason prefer a lower level implementation, read the same section in the *Advanced usage (low-level)* chapter.

6.2 Command line usage

It's possible to generate a signed URL from command line using the `ska.generate_signed_url` module.

Arguments

```
-h, --help            show this help message and exit

-au AUTH_USER, --auth-user AUTH_USER
                      `auth_user` value

-sk SECRET_KEY, --secret-key SECRET_KEY
                      `secret_key` value

-vu VALID_UNTIL, --valid-until VALID_UNTIL
                      `valid_until` value

-l LIFETIME, --lifetime LIFETIME
                      `lifetime` value

-u URL, --url URL     URL to sign

-sp SIGNATURE_PARAM, --signature-param SIGNATURE_PARAM
                      (GET) param holding the `signature` value

-aup AUTH_USER_PARAM, --auth-user-param AUTH_USER_PARAM
                      (GET) param holding the `auth_user` value

-vup VALID_UNTIL_PARAM, --valid-until-param VALID_UNTIL_PARAM
                      (GET) param holding the `auth_user` value
```

Example

```
ska-sign-url -au user -sk your-secret-key --url http://example.com
```

6.3 Advanced usage (low-level)

6.3.1 Sender side

Required imports.

```
from ska import Signature, RequestHelper
```

Generate a signature.

```
signature = Signature.generate_signature(
    auth_user='user',
    secret_key='your-secret-key'
)
```

Default lifetime of a signature is 10 minutes (600 seconds). If you want it to be different, provide a `lifetime` argument to `generate_signature` method.

```
signature = Signature.generate_signature(
    auth_user='user',
    secret_key='your-secret-key',
    lifetime=120  # Signature lifetime set to 120 seconds.
)
```

Adding of additional data to the signature works in the same way as in `sign_url`.

```
signature = Signature.generate_signature(
    auth_user='user',
    secret_key='your-secret-key',
    extra={
        'email': 'doe@example.com',
        'last_name': 'Doe',
        'first_name': 'Joe'
    }
)
```

For HMAC SHA-384 algorithm it would look as follows.

```
from ska import HMACSHA384Signature

signature = HMACSHA384Signature.generate_signature(
    auth_user='user',
    secret_key='your-secret-key'
)
```

Your endpoint operates with certain param names and you need to wrap generated signature params into the URL. In order to have the job done in an easy way, create a request helper. Feed names of the (GET) params to the request helper and let it make a signed endpoint URL for you.

```
request_helper = RequestHelper(
    signature_param='signature',
    auth_user_param='auth_user',
    valid_until_param='valid_until'
)
```

Append signature params to the endpoint URL.

```
signed_url = request_helper.signature_to_url(
    signature=signature,
    endpoint_url='http://e.com/api/'
)
```

```
GET http://e.com/api/?valid_until=1378045287.0&auth_user=user&
↪signature=YlZpLFsjUKBalL4x5trhkeEgqE8%3D
```

Make a request.

```
import requests
r = requests.get(signed_url)
```

For HMAC SHA-384 algorithm it would look as follows.

```
from ska import HMACSHA384Signature

request_helper = RequestHelper(
    signature_param='signature',
    auth_user_param='auth_user',
    valid_until_param='valid_until',
    signature_cls=HMACSHA384Signature
)

signed_url = request_helper.signature_to_url(
    signature=signature,
    endpoint_url='http://e.com/api/'
)
```

6.3.2 Recipient side

Required imports.

```
from ska import RequestHelper
```

Create a request helper. Your endpoint operates with certain param names. In order to have the job done in an easy way, we feed those params to the request helper and let it extract data from signed request for us.

```
request_helper = RequestHelper(
    signature_param='signature',
    auth_user_param='auth_user',
    valid_until_param='valid_until'
)
```

Validate the request data. Note, that `request.GET` is given just as an example.

```
validation_result = request_helper.validate_request_data(
    data=request.GET,
    secret_key='your-secret-key'
)
```

Your implementation further depends on you, but may look as follows.

```
if validation_result.result:
    # Validated, proceed further
    # ...
else:
    # Validation not passed.
    raise Http404(validation_result.reason)
```

You can also just validate the signature by calling `validate_signature` method of the `ska.Signature`.

```
Signature.validate_signature(
    signature='EBS6ipiqRLa6TY5vxIvZU30FpnM=',
    auth_user='user',
    secret_key='your-secret-key',
    valid_until='1377997396.0'
)
```

6.4 Django integration

`ska` comes with Django model- and view-decorators for producing signed URLs and and validating the endpoints, as well as with authentication backend, which allows password-less login into Django web site using `ska` generated signature tokens. There's also a template tag for signing URLs.

6.4.1 Demo

In order to be able to quickly evaluate the `ska`, a demo app (with a quick installer) has been created (works on Ubuntu/Debian, may work on other Linux systems as well, although not guaranteed). Follow the instructions below for having the demo running within a minute.

Grab the latest `ska_example_app_installer.sh` and execute it:

```
wget -O - https://raw.github.com/barseghyanartur/ska/stable/examples/ska_example_app_
↪installer.sh | bash
```

Open your browser and test the app.

Foo listing (ska protected views):

- URL: http://127.0.0.1:8001/foo/

Authentication page (ska authentication backend):

- URL: http://127.0.0.1:8001/foo/authenticate/

Django admin interface:

- URL: http://127.0.0.1:8001/admin/
- Admin username: test_admin
- Admin password: test

6.4.2 Configuration

Secret key (`str`) must be defined in `settings` module of your project.

```
SKA_SECRET_KEY = 'my-secret-key'
```

The following variables can be overridden in `settings` module of your project.

- `SKA_UNAUTHORISED_REQUEST_ERROR_MESSAGE` (`str`): Plain text error message. Defaults to "Unauthorised request. {0}".
- `SKA_UNAUTHORISED_REQUEST_ERROR_TEMPLATE` (`str`): Path to 401 template that should be rendered in case of 401 responses. Defaults to empty string (not provided).
- `SKA_AUTH_USER` (`str`): The `auth_user` argument for `ska.sign_url` function. Defaults to "ska-auth-user".

See the working example project.

6.4.3 Multiple secret keys

Imagine, you have a site to which you want to offer a password-less login for various clients/senders and you don't want them all to have one shared secret key, but rather have their own one. Moreover, you specifically want to execute very custom callbacks not only for each separate client/sender, but also for different sort of users authenticating.

```
                          ________________
                         |                 |
                         | Site providing  |
                         | authentication  |
                         | ______________  |
                         |                 |
                         | custom secret   |
                         |    keys per     |
                         |     client      |
                         | ______________  |
                         |                 |
                         | Site 1: 'sk-1'  |
             ----------->| Site 2: 'sk-2' |<-----------
             |           | Site 3: 'sk-3'  |            |
             |    ---->| Site 4: 'sk-4' |<----       |
             |    |      |________________|     |      |
             |    |                              |      |
             |    |                               |      |
_____________   _____________   _____________   _____________
| Site 1      | | Site 2      | | Site 3      | | Site 4      |
| ___________ | | ___________ | | ___________ | | ___________ |
| secret key  | | secret key  | | secret key  | | secret key  |
|   'sk-1'    | |   'sk-2'    | |   'sk-3'    | |   'sk-4'    |
|_____________| |_____________| |_____________| |_____________|
```

In order to make the stated above possible, the concept of providers is introduced. You can define a secret key, callbacks or redirect URL. See an example below. Note, that keys of the `SKA_PROVIDERS` ("client_1", "client_2", etc.) are the provider keys.

```
SKA_PROVIDERS = {
    # ********************************************************
    # ******************** Basic gradation *******************
    # ********************************************************
    # Site 1
```

(continues on next page)

(continued from previous page)

```
    'client_1': {
        'SECRET_KEY': 'sk-1',
    },

    # Site 2
    'client_2': {
        'SECRET_KEY': 'sk-2',
    },

    # Site 3
    'client_3': {
        'SECRET_KEY': 'sk-3',
    },

    # Site 4
    'client_4': {
        'SECRET_KEY': 'sk-4',
    },

    # ********************************************************
    # ******* You make gradation as complex as you wish ******
    # ********************************************************
    # Client 1, group users
    'client_1.users': {
        'SECRET_KEY': 'client-1-users-secret-key',
    },

    # Client 1, group power_users
    'client_1.power_users': {
        'SECRET_KEY': 'client-1-power-users-secret-key',
        'USER_CREATE_CALLBACK': 'foo.ska_callbacks.client1_power_users_create',
    },

    # Client 1, group admins
    'client_1.admins': {
        'SECRET_KEY': 'client-1-admins-secret-key',
        'USER_CREATE_CALLBACK': 'foo.ska_callbacks.client1_admins_create',
        'REDIRECT_AFTER_LOGIN': '/admin/'
    },
}
```

See the *Callbacks* section for the list of callbacks. Note, that callbacks defined in the `SKA_PROVIDERS` are overrides. If a certain callback isn't defined in the `SKA_PROVIDERS`, authentication backend falls back to the respective default callback function.

Obviously, server would have to have the full list of providers defined. On the client side you would only have to store the general secret key and of course the provider UID(s).

When making a signed URL on the sender side, you should be providing the `provider` key in the `extra` argument. See the example below for how you would do it for `client_1.power_users`.

```
from ska import sign_url
from ska.defaults import DEFAULT_PROVIDER_PARAM
```

(continues on next page)

(continued from previous page)

```
server_ska_login_url = 'https://server-url.com/ska/login/'

signed_remote_ska_login_url = sign_url(
    auth_user='test_ska_user',
    # Using provider-specific secret key. This value shall be equal to
    # the value of SKA_PROVIDERS['client_1.power_users']['SECRET_KEY'],
    # defined in your projects' Django settings module.
    secret_key='client-1-power-users-secret-key',
    url=server_ska_login_url,
    extra={
        'email': 'test_ska_user@mail.example.com',
        'first_name': 'John',
        'last_name': 'Doe',
        # Using provider specific string. This value shall be equal to
        # the key string "client_1.power_users" of SKA_PROVIDERS,
        # defined in your projcts' Django settings module.
        DEFAULT_PROVIDER_PARAM: 'client_1.power_users',
    }
)
```

6.4.4 Django model method decorator `sign_url`

This is most likely be used in module `models` (models.py).

Imagine, you have a some objects listing and you want to protect the URLs to be viewed by authorised parties only. You would then use `get_signed_absolute_url` method when rendering the listing (HTML).

```
from django.db import models
from django.utils.translation import ugettext_lazy as _
from django.core.urlresolvers import reverse

from ska.contrib.django.ska.decorators import sign_url


class FooItem(models.Model):

    title = models.CharField(_("Title"), max_length=100)
    slug = models.SlugField(unique=True, verbose_name=_("Slug"))
    body = models.TextField(_("Body"))

    # Unsigned absolute URL, which goes to the foo item detail page.
    def get_absolute_url(self):
        return reverse('foo.detail', kwargs={'slug': self.slug})

    # Signed absolute URL, which goes to the foo item detail page.
    @sign_url()
    def get_signed_absolute_url(self):
        return reverse('foo.detail', kwargs={'slug': self.slug})
```

Note, that `sign_url` decorator accepts the following optional arguments.

- `auth_user` (`str`): Username of the user making the request.

- `secret_key`: The shared secret key. If set, overrides the `SKA_SECRET_KEY` variable set in the `settings` module of your project.
- `valid_until` (`float` or `str`): Unix timestamp. If not given, generated automatically (now + lifetime).
- `lifetime` (`int`): Signature lifetime in seconds.
- `suffix` (`str`): Suffix to add after the `endpoint_url` and before the appended signature params.
- `signature_param` (`str`): Name of the GET param name which would hold the generated signature value.
- `auth_user_param` (`str`): Name of the GET param name which would hold the `auth_user` value.
- `valid_until_param` (`str`): Name of the GET param name which would hold the `valid_until` value.

6.4.5 Django view decorator `validate_signed_request`

To be used to protect views (file views.py). Should be applied to views (endpoints) that require signed requests. If checks are not successful, a `ska.contrib.django.ska.http.HttpResponseUnauthorized` is returned, which is a subclass of Django's `django.http.HttpResponse`. You can provide your own template for 401 error. Simply point the `SKA_UNAUTHORISED_REQUEST_ERROR_TEMPLATE` in `settings` module to the right template. See `ska/contrib/django/ska/templates/ska/401.html` as a template example.

```python
from ska.contrib.django.ska.decorators import validate_signed_request

# Your view that shall be protected
@validate_signed_request()
def detail(request, slug, template_name='foo/detail.html'):
    # Your code
```

Note, that `validate_signed_request` decorator accepts the following optional arguments.

- `secret_key` (`str`) : The shared secret key. If set, overrides the `SKA_SECRET_KEY` variable set in the `settings` module of your project.
- `signature_param` (`str`): Name of the (for example GET or POST) param name which holds the `signature` value.
- `auth_user_param` (`str`): Name of the (for example GET or POST) param name which holds the `auth_user` value.
- `valid_until_param` (`str`): Name of the (foe example GET or POST) param name which holds the `valid_until` value.

If you're using class based views, use the `m_validate_signed_request` decorator instead of `validate_signed_request`.

6.4.6 Template tags

There are two template tags modules: `ska_tags` and `ska_constance_tags`. They are functionally identical, although `ska_constance_tags` is tied to `django-constance`.

For standard settings configurations, template tags shall be loaded as follows:

```
{% load ska_tags %}
```

For `django-constance` based settings configurations, template tags shall be loaded as follows:

```
{% load ska_constance_tags %}
```

Note, that if you want to use `ska_constance_tags`, add the `ska.contrib.django.ska.integration.constance_integration` line to your``INSTALLED_APPS``:

```
INSTALLED_APPS = (
    # ...
    'ska.contrib.django.ska.integration.constance_integration',
    # ...
)
```

6.4.6.1 sign_url

The `sign_url` template tag accepts template context and the following params:

- `url`
- `auth_user`: If not given, `request.user.get_username()` is used.
- `secret_key`: If not given, the secret key from settings is used.
- `valid_until`: If not given, calculated from `lifetime`.
- `lifetime`: Defaults to `ska.defaults.SIGNATURE_LIFETIME`.
- `suffix`: Defaults to `ska.defaults.DEFAULT_URL_SUFFIX`.
- `signature_param`: Defaults to `ska.defaultsDEFAULT_SIGNATURE_PARAM`.
- `auth_user_param`: Defaults to `ska.defaults.DEFAULT_AUTH_USER_PARAM`.
- `valid_until_param`: Defaults to `ska.defaults.DEFAULT_VALID_UNTIL_PARAM`.
- `signature_cls`: Defaults to `ska.signatures.Signature`.

Usage example:

```
{% load ska_tags %}

{% for item in items%}

    {% sign_url item.get_absolute_url as item_signed_absolute_url %}
    <a href="{{ item_signed_absolute_url }}">{{ item }}</a>

{% endfor %}
```

6.4.6.2 provider_sign_url

The `provider_sign_url` template tag accepts template context and the following params:

- `url`
- `provider`: Provider name.
- `auth_user`: If not given, `request.user.get_username()` is used.
- `valid_until`: If not given, calculated from `lifetime`.
- `lifetime`: Defaults to `ska.defaults.SIGNATURE_LIFETIME`.

- `suffix`: Defaults to `ska.defaults.DEFAULT_URL_SUFFIX`.
- `signature_param`: Defaults to `ska.defaultsDEFAULT_SIGNATURE_PARAM`.
- `auth_user_param`: Defaults to `ska.defaults.DEFAULT_AUTH_USER_PARAM`.
- `valid_until_param`: Defaults to `ska.defaults.DEFAULT_VALID_UNTIL_PARAM`.
- `signature_cls`: Defaults to `ska.signatures.Signature`.
- `fail_silently`: Defaults to False.

Usage example:

```
{% load ska_tags %}

{% for item in items%}

    {% provider_sign_url url=item.get_absolute_url provider='client_1.users' as item_
↪signed_absolute_url %}
    <a href="{{ item_signed_absolute_url }}">{{ item }}</a>

{% endfor %}
```

6.4.7 Authentication backends

Allows you to get a password-less login to Django web site.

At the moment there are two backends implemented:

- *SkaAuthenticationBackend*: Uses standard Django settings.
- *SkaAuthenticationConstanceBackend*: Relies on dynamic settings functionality provided by `django-constance`.

By default, number of logins using the same token is not limited. If you wish that single tokens become invalid after first use, set the following variables to True in your projects' Django settings module.

```
SKA_DB_STORE_SIGNATURES = True
SKA_DB_PERFORM_SIGNATURE_CHECK = True
```

6.4.7.1 SkaAuthenticationBackend

`SkaAuthenticationBackend` uses standard Django settings.

6.4.7.1.1 Recipient side

Recipient is the host (Django site), to which the sender tries to get authenticated (log in). On the recipient side the following shall be present.

6.4.7.1.1.1 settings.py

```
AUTHENTICATION_BACKENDS = (
    'ska.contrib.django.ska.backends.SkaAuthenticationBackend',
    'django.contrib.auth.backends.ModelBackend',
)

INSTALLED_APPS = (
    # ...
    'ska.contrib.django.ska',
    # ...
)

SKA_SECRET_KEY = 'secret-key'
SKA_UNAUTHORISED_REQUEST_ERROR_TEMPLATE = 'ska/401.html'
SKA_REDIRECT_AFTER_LOGIN = '/foo/logged-in/'
```

6.4.7.1.1.2 urls.py

```
urlpatterns = [
    url(r'^ska/', include('ska.contrib.django.ska.urls')),
    url(r'^admin/', include(admin.site.urls)),
]
```

6.4.7.1.1.3 Callbacks

There are several callbacks implemented for authentication backend.

- `USER_VALIDATE_CALLBACK` (`str`): Validate request callback. Created to allow adding custom logic to the incoming authentication requests. The main purpose is to provide a flexible way of raising exceptions if the incoming authentication request shall be blocked (for instance, email or username is in black-list or right the opposite - not in the white list). The only aim of the `USER_VALIDATE_CALLBACK` is to raise a `django.core.PermissionDenied` exception if request data is invalid. In that case authentication flow will halt. All other exceptions would simply be ignored (but logged) and if no exception raised, the normal flow would be continued.
- `USER_GET_CALLBACK` (`str`): Fired if user was successfully fetched from database (existing user).
- `USER_CREATE_CALLBACK` (`str`): Fired right after user has been created (user didn't exist).
- `USER_INFO_CALLBACK` (`str`): Fired upon successful authentication.

Example of a callback function (let's say, it resides in module `my_app.ska_callbacks`):

```
def my_callback(user, request, signed_request_data)
    # Your code
```

…where:

- `user` is `django.contrib.auth.models.User` instance.
- `request` is `django.http.HttpRequest` instance.
- `signed_request_data` is dictionary with signed request data.

For example, if you need to assign user to some local Django group, you could specify the group name on the client side (add it to the `extra` dictionary) and based on that, add the user to the group in the callback.

The callback is a path qualifier of the callback function. Considering the example above, it would be `my_app.ska_callbacks.my_callback`.

Prefix names of each callback variable with `SKA_` in your projects' settings module.

Example:

```
SKA_USER_GET_CALLBACK = 'my_app.ska_callbacks.my_get_callback'
SKA_USER_CREATE_CALLBACK = 'my_app.ska_callbacks.my_create_callback'
```

6.4.7.1.2 Sender side

Sender is the host (another Django web site) from which users authenticate to the Recipient using signed URLs.

On the sender side, the only thing necessary to be present is the `ska` module for Django and of course the same `SECRET_KEY` as on the server side. Further, the server `ska` login URL (in our case "/ska/login/") shall be signed using `ska` (for example, using `sign_url` function). The `auth_user` param would be used as a Django username. See the example below.

```
from ska import sign_url
from ska.contrib.django.ska.settings import SECRET_KEY

server_ska_login_url = 'https://server-url.com/ska/login/'

signed_url = sign_url(
    auth_user='test_ska_user_0',
    secret_key=SECRET_KEY,
    url=server_ska_login_url,
    extra={
        'email': 'john.doe@mail.example.com',
        'first_name': 'John',
        'last_name': 'Doe',
    }
)
```

Note, that you `extra` dictionary is optional! If `email`, `first_name` and `last_name` keys are present, upon successful validation, the data would be saved into users' profile.

Put this code, for instance, in your view and then make the generated URL available in template context and render it as a URL so that user can click on it for authenticating to the server.

```
def auth_to_server(request, template_name='auth_to_server.html'):
    # Some code + obtaining the `signed_url` (code shown above)
    context = {'signed_url': signed_url}

    return render(request, template_name, context)
```

6.4.7.2 SkaAuthenticationConstanceBackend

Relies on dynamic settings functionality provided by django-constance.

Only differences with `SkaAuthenticationBackend` are mentioned.

Note: Additional requirements shall be installed. See the constance.txt file for additional requirements (`django-constance`, `django-json-widget`, `django-picklefield`, `jsonfield2` and `redis`).

6.4.7.2.1 settings.py

```
AUTHENTICATION_BACKENDS = (
    'ska.contrib.django.ska.backends.constance_backend.SkaAuthenticationConstanceBackend
↪',
    'django.contrib.auth.backends.ModelBackend',
)

INSTALLED_APPS = (
    # ...
    'constance',  # django-constance
    'ska.contrib.django.ska',
    'django_json_widget',  # For nice admin JSON widget
    # ...
)

CONSTANCE_CONFIG = {
    'SKA_PROVIDERS': (
        {},  # The default value
        'JSON data',  # Help text in admin
        'JSONField_config',  # Field config
    )
}

CONSTANCE_ADDITIONAL_FIELDS = {
    'JSONField_config': [
        # `jsonfield2` package might be used for storing the JSON field,
        # however, at the moment of writing it has a bug which makes
        # the JSON invalid after the first save. To avoid that, it has
        # been patched and resides in examples/simple/jsonfield2_addons/
        # module.
        'jsonfield2_addons.forms.JSONField',
        {
            'widget': 'django_json_widget.widgets.JSONEditorWidget',
        }
    ],
}

CONSTANCE_BACKEND = 'constance.backends.redisd.RedisBackend'

CONSTANCE_REDIS_CONNECTION = {
    'host': 'localhost',
```

(continues on next page)

(continued from previous page)

```
    'port': 6379,
    'db': 0,
}
```

Note: In very tiny bits, although not required, the jsonfield2 and django-json-widget packages are used for editing of the SKA_PROVIDERS setting in Django admin.

Note: In the example shown above, the RedisBackend of django-constance is used. You could also use DatabaseBackend. Study the documentation for more.

Note: If your *SKA_PROVIDERS* settings are stored in the constance as str instead of dict, set the setting SKA_CONSTANCE_SETTINGS_PARSE_FROM_JSON to True.

With DatabaseBackend it would look as follows:

```
CONSTANCE_BACKEND = 'constance.backends.database.DatabaseBackend'

INSTALLED_APPS = (
    # ...
    'constance.backends.database',
    # ...
)
```

Quick demo of the dynamic backend

- Clone this project:

```
git clone git@github.com:barseghyanartur/ska.git
```

- Install/migrate:

```
./scripts/install.sh
pip install -r examples/requirements/django_2_1.txt
./scripts/migrate.sh --settings=settings.constance_settings
```

- Run:

```
./scripts/runserver.sh --settings=settings.constance_settings
```

- Go to http://localhost:8000/admin/constance/config/.
- Paste the following code:

```
{
   "client_1.users":{
      "SECRET_KEY":"client-1-users-secret-key"
   },
   "client_1.power_users":{
      "SECRET_KEY":"client-1-power-users-secret-key",
```

(continues on next page)

(continued from previous page)

```
      "USER_CREATE_CALLBACK":"foo.ska_callbacks.client1_power_users_create"
   },
   "client_1.admins":{
      "SECRET_KEY":"client-1-admins-secret-key",
      "USER_CREATE_CALLBACK":"foo.ska_callbacks.client1_admins_create",
      "USER_GET_CALLBACK":"foo.ska_callbacks.client1_admins_get",
      "USER_INFO_CALLBACK":"foo.ska_callbacks.client1_admins_info_constance",
      "REDIRECT_AFTER_LOGIN":"/admin/auth/user/"
   }
}
```

- Open http://localhost:8000/foo/authenticate/ in another browser and navigate to the `Log in - client_1.admins` link in the `Success` table column of the `By provider` section. Upon clicking, you should be logged in. You have used the dynamic settings.

6.4.7.2.2 urls.py

`django-constance` specific views and urls are used. See ska.contrib.django.ska.views.constance_views and ska.contrib.django.ska.urls.constance_urls for the reference.

```
urlpatterns = [
    url(r'^ska/', include('ska.contrib.django.ska.urls.constance_urls')),
    url(r'^admin/', include(admin.site.urls)),
]
```

6.4.7.3 Custom authentication backend

To implement alternative authentication backend, see the following example:

```
from constance import config

from ska.contrib.django.backends import BaseSkaAuthenticationBackend

class SkaAuthenticationConstanceBackend(BaseSkaAuthenticationBackend):
    """Authentication backend."""

    def get_settings(self):
        """

        :return:
        """
        return config.SKA_PROVIDERS
```

That's it. The only thing the `get_settings` method shall return is `dict` with providers data (see the *Multiple secret keys* for the reference; return value of the `get_settings` is ``SKA_PROVIDERS` dict).

6.4.7.4 Purging of old signature data

If you have lots of visitors and the SKA_DB_STORE_SIGNATURES set to True, your database grows. If you wish to get rid of old signature token data, you may want to execute the following command using a cron job.

```
./manage.py ska_purge_stored_signature_data
```

6.4.7.5 Security notes

From point of security, you should be serving the following pages via HTTP secure connection:

- The server login page (/ska/login/).
- The client page containing the authentication links.

6.4.8 Django REST Framework integration

6.4.8.1 Permission classes

For protecting views without actually being authenticated into the system, specific permission classes are implemented (for both plan settings and provider settings, as well as both plain- and provider-settings work in combination with `django-constance` package).

The following permission classes are implemented:

- `SignedRequestRequired`
- `ProviderSignedRequestRequired`
- `ConstanceSignedRequestRequired`
- `ConstanceProviderSignedRequestRequired`

ProviderSignedRequestRequired example

```python
from rest_framework.viewsets import ModelViewSet

from ska.contrib.django.ska.integration.drf.permissions import (
    ProviderSignedRequestRequired
)

from .models import FooItem
from .serializers import FooItemSerializer

class FooItemViewSet(ModelViewSet):
    """FooItem model viewset."""

    permission_classes = (ProviderSignedRequestRequired,)
    queryset = FooItem.objects.all()
    serializer_class = FooItemSerializer
```

Signing requests

Requests are signed the same way. Sample code:

```
# Given that we have `auth_user`, `auth_user_email`, `provider_name`
# (and the rest), the code would look as follows:

from ska import sign_url
from ska.defaults import DEFAULT_PROVIDER_PARAM

extra = {
    'email': auth_user_email,
    'first_name': first_name,
    'last_name': last_name,
}

if provider_name:
    extra.update({DEFAULT_PROVIDER_PARAM: provider_name})

signed_url = sign_url(
    auth_user=auth_user,
    secret_key=secret_key,
    url=url,
    extra=extra
)
```

6.4.8.2 JWT tokens for authentication

For obtaining JWT tokens for authentication. Also works with `django-constance`.

settings example

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}
```

urls example

```
urlpatterns = [
    # ...
    url(
        r'^ska-rest/',
        include('ska.contrib.django.ska.integration.drf.urls.jwt_token')
    ),
]
```

Sample request

```
http://localhost:8008/ska-rest/obtain-jwt-token/
    ?signature=P92KWDDe0U84Alvu0tvmYoi8e8s%3D
    &auth_user=test_ska_user
    &valid_until=1548195246.0
    &extra=email%2Cfirst_name%2Clast_name
```

(continues on next page)

(continued from previous page)

```
    &email=test_ska_user%40mail.example.com
    &first_name=John
    &last_name=Doe
```

Sample response

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```

```
{
    "token": "eyJ0eXAiO.eyJ1c2VyX2lkIjo.m_saOvyKBO3"
}
```

CHAPTER SEVEN

TESTING

Simply type:

```
pytest
```

Or use tox:

```
tox
```

Or use tox to check specific env:

```
tox -e py39
```

Or run Django tests:

```
python examples/simple/manage.py test ska --settings=settings.testing
```

CHAPTER

EIGHT

WRITING DOCUMENTATION

Keep the following hierarchy.

```
=====
title
=====

header
======

sub-header
----------

sub-sub-header
~~~~~~~~~~~~~~

sub-sub-sub-header
++++++++++++++++++

sub-sub-sub-sub-header
^^^^^^^^^^^^^^^^^^^^^^

sub-sub-sub-sub-sub-header
**************************
```

CHAPTER NINE

LICENSE

GPL-2.0-only OR LGPL-2.1-or-later

CHAPTER TEN

SUPPORT

For security issues contact me at the e-mail given in the *Author* section.

For overall issues, go to GitHub.

CHAPTER ELEVEN

AUTHOR

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER

TWELVE

PROJECT DOCUMENTATION

Contents:

Table of Contents

12.1 Release history and notes

Sequence based identifiers are used for versioning (schema follows below):

```
major.minor[.revision]
```

- It's always safe to upgrade within the same minor version (for example, from 0.3 to 0.3.4).
- Minor version changes might be backwards incompatible. Read the release notes carefully before upgrading (for example, when upgrading from 0.3.4 to 0.4).
- All backwards incompatible changes are mentioned in this document.

12.1.1 1.10

2023-08-27

- Tested against Python 3.11.
- Mark *django-nine* as optional dependency.
- Drop support for Python < 3.7.
- Drop support for Django < 3.2 and 4.0.
- Tested against Django 4.1 and 4.2.
- Upgrade relevant contrib code to support both `django-constance` >= 2.8.x and 3.x.

12.1.2 1.9.1

2021-11-18

- Tested against Python 3.10.

12.1.3 1.9

2021-08-18

- Add *value_dumper* to most of the functions/methods related to signature generation/validation. It's aimed to make signatures generated in languages better compatible with *ska*.
- Add *quoter* to most of the functions/methods related to signature generation/validation. It's aimed to make signatures generated in languages better compatible with *ska*.

12.1.4 1.8.2

2021-06-18

- Add typing to most of the code parts.

12.1.5 1.8.1

2021-06-10

- Wipe out old flavour from code.
- Blackify.

12.1.6 1.8

2021-06-10

Additions to the Django contrib app

- Drop Python 2.7 and 3.5 support.
- Tested against Django 2.2, 3.0, 3.1 and 3.2.
- Tested against Python 3.8 and 3.9.

- `django-constance` specific template tags have been moved to `ska.contrib.django.ska.integration.constance_integration`. Update your Django settings accordingly.
- `django-constance` specific authentication backend has been moved to `'ska.contrib.django.ska.backends.constance_backend.SkaAuthenticationConstanceBackend`. Update your Django settings accordingly.
- `django-constance` specific DRF permission classes (`ConstanceSignedRequestRequired` and `ConstanceProviderSignedRequestRequired`) have been moved to `ska.contrib.django.ska.integration.drf.permissions.constance_permissions`. Update your Django settings accordingly.

12.1.7 1.7.5

2019-05-15

- Fixes in ska-sign-url on Python 3.5.

12.1.8 1.7.4

2019-05-12

Minor additions to the Django contrib app

- Introduce `SKA_CONSTANCE_SETTINGS_PARSE_FROM_JSON` directive for parsing the data stored in `django-constance` (instead of treating it as `dict`). Default value is `False`.

12.1.9 1.7.3

2019-03-13

Fixes in the Django contrib app

- Handle cases when *request* is not passed to the authentication backend.

12.1.10 1.7.2

2019-02-23

Additions to the Django contrib app

- Added *provider_sign_url* template tag to the existing *ska_tags* template tags module.
- Added a new *ska_constance_tags* template tags module (to be used in combination with *django-constance*).

12.1.11 1.7.1

2019-01-22

Additions to the Django contrib app

- Added Django REST framework JWT token obtain view (for authentication).
- Fixes in the authentication backend *SkaAuthenticationConstanceBackend*.

12.1.12 1.7

2018-12-28

Additions to the Django contrib app

- Added Django REST framework integration (for signing ViewSets).

12.1.13 1.6.12

2018-12-25

Additions to the Django contrib app

- Added additional callback `USER_VALIDATE_CALLBACK` to the authentication backends which is fired right after the signature validation to allow custom validation logic for the incoming authentication requests.

12.1.14 1.6.11

2018-12-20

Additions to the Django contrib app

- Authentication backend has been made customisable. Most of the code is moved to the `BaseSkaAuthenticationBackend`. Introduced new authentication backend `SkaAuthenticationConstanceBackend` to be used in combination with `django-constance`.

12.1.15 1.6.10

2018-12-16

Additions to the Django contrib app

- Fixes in the callbacks import of the Django contrib app.
- Testing shell commands; minor fixes in tests.

12.1.16 1.6.9

2018-12-07

- Tested against Python 3.7.
- Add initial migrations for Django contrib package.

12.1.17 1.6.8

2018-12-03

Note: Release dedicated to Charles Aznavour. Rest in peace, maestro.

- Django 2.0 and 2.1 compatibility.
- Upgrade test suite.

- Fixes in docs.
- Python 3.4 is removed from support matrix (however, it might still work).

12.1.18 1.6.7

2017-02-09

- Tested against Python 3.6 and Django 1.11 (alpha).

12.1.19 1.6.6

2016-12-21

- Minor fixes.

12.1.20 1.6.5

2016-12-06

- Fixed in docs.

12.1.21 1.6.4

2016-12-06

- Added template tags library for Django integration.

12.1.22 1.6.3

2016-12-04

- Fixes in django ska decorators.

12.1.23 1.6.2

2016-12-03

- Fixed broken example installer.

12.1.24 1.6.1

2016-12-03

- Fixes in tests of django-ska package.
- Add shell.py command for easy debugging.
- Minor fixes.
- Clean up docs.

12.1.25 1.6

2016-12-02

- Django 1.8, 1.9 and 1.10 compatibility.
- pep8 fixes.
- The `six` package requirement increased to six >= 1.9.
- Drop support of Django < 1.8 (it still may work, but no longer guaranteed).
- Drop support of Python 2.6.x.
- Fix broken Django authentication backend, due to deprecation of `request.REQUEST`.

12.1.26 1.5

2014-06-04

- Introducing abstract signature class in order to make it possible to define more hash algorithms.
- Added HMAC MD5, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384 and HMAC SHA-512 hash algorythms. HMAC SHA-1 remains a default.

12.1.27 1.4.4

2014-05-06

- Add `ska-sign-url` terminal command (Linux only).

12.1.28 1.4.3

2014-02-28

- The `ValidationResult` class is slightly changed. The `reason` property is replaced with `errors` (while `reason` is left mainly for backwards compatibility). For getting human readable message you're encouraged to use the `message` property (string) instead of joining strings manually. Additionally, each error got a separate object (see `error_codes` module): `INVALID_SIGNATURE` and `SIGNATURE_TIMESTAMP_EXPIRED`.
- Minor documentation improvements.

12.1.29 1.4.2

2013-12-25

- Minor fixes.
- Added authentication backend tests.
- Added tumpering tests.
- Minor documentation improvements.

12.1.30 1.4.1

2013-12-23

- Armenian, Dutch and Russian translations added for Django app.
- Documentation improved.

12.1.31 1.4

2013-12-21

- Providers concept implemented. It's now possible to handle multiple secret keys and define custom callbacks and redirect URLs per provider. See the docs for more.
- Better example project.
- Better documentation.

12.1.32 1.3

2013-12-21

- Make it possible to add additional data to the signed request by providing an additional `extra` argument.
- Reflect the new functionality in Django app.
- Better documentation.

12.1.33 1.2

2013-12-17

- Optionally storing the authentication tokens into the database, when used with Django auth backend.
- Optionally checking, if signature token has already been used to log into Django. If so, ignoring the login attempt. A management command is added to purge old signature data.
- Demo (quick installer) added.

12.1.34 1.1

2013-12-14

- Class based views validation decorator added.
- Authentication backend for Django based on authentication tokens generated with `ska`.

12.1.35 1.0

2013-12-13

- Lowered `six` version requirement to 1.1.0.

12.1.36 0.9

2013-10-16

- Lowered `six` version requirement to 1.4.0.

12.1.37 0.8

2013-10-12

- Contrib package `ska.contrib.django.ska` added for better Django integration.

12.1.38 0.7

2013-09-12

- Pinned version requirement of `six` package to 1.4.1.

12.1.39 0.6

2013-09-06

- Python 2.6.8 and 3.3 support addeded.

12.1.40 0.5

2013-09-05

- Stable release.

12.1.41 0.4

2013-09-04

- Adding shortcuts for handling dictionaries.
- Improved documentation.

12.1.42 0.3

2013-09-04

- Adding commands to generate the URLs.

12.1.43 0.2

2013-09-02

- Fixed docs.

12.1.44 0.1

2013-09-01

- Initial beta release.

12.2 Security Policy

12.2.1 Reporting a Vulnerability

Do not report security issues on GitHub!

Please report security issues by emailing Artur Barseghyan <artur.barseghyan@gmail.com>.

12.2.2 Supported Versions

Make sure to use the latest version.

The two most recent `ska` release series receive security support.

For example, during the development cycle leading to the release of `ska` 1.10.x, support will be provided for `ska` 1.9.x.

Upon the release of `ska` 1.11, security support for `ska` 1.9.x will end.

Version	Supported
1.10.x	Yes
1.9.x	Yes
< 1.9	No

12.3 Contributor Covenant Code of Conduct

12.3.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

12.3.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

12.3.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

12.3.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

12.3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at artur.barseghyan@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

12.3.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

12.3.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

12.3.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

12.3.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

12.3.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

12.3.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

12.4 Contributor guidelines

12.4.1 Developer prerequisites

12.4.1.1 pre-commit

Refer to pre-commit for installation instructions.

TL;DR:

```
pip install pipx --user  # Install pipx
pipx install pre-commit  # Install pre-commit
pre-commit install  # Install pre-commit hooks
```

Installing pre-commit will ensure you adhere to the project code quality standards.

12.4.2 Code standards

black, isort, ruff and doc8 will be automatically triggered by pre-commit. Still, if you want to run checks manually:

```
./scripts/black.sh
./scripts/doc8.sh
./scripts/isort.sh
./scripts/ruff.sh
```

12.4.3 Requirements

Requirements are compiled using pip-tools.

```
./scripts/compile_requirements.sh
```

12.4.4 Virtual environment

You are advised to work in virtual environment.

TL;DR:

```
python -m venv env
pip install -e .
pip install -r examples/requirements/django_4_2.txt
```

12.4.5 Documentation

Check documentation.

12.4.6 Testing

Check testing.

If you introduce changes or fixes, make sure to test them locally using all supported environments. For that use tox.

```
tox
```

In any case, GitHub Actions will catch potential errors, but using tox speeds things up.

12.4.7 Pull requests

You can contribute to the project by making a pull request.

For example:

- To fix documentation typos.
- To improve documentation (for instance, to add new recipe or fix an existing recipe that doesn't seem to work).
- To introduce a new feature (for instance, add support for a non-supported file type).

Good to know:

- Test suite makes extensive use of parametrization. Make sure you have added your changes in the right place.

General list to go through:

- Does your change require documentation update?
- Does your change require update to tests?
- Did you test both Latin and Unicode characters?
- Does your change rely on third-party cloud based service? If so, please make sure it's added to tests that should be retried a couple of times. Example: `@pytest.mark.flaky(reruns=5)`.

When fixing bugs (in addition to the general list):

- Make sure to add regression tests.

When adding a new feature (in addition to the general list):

- Check the licenses of added dependencies carefully and make sure to list them in prerequisites.
- Make sure to update the documentation (check whether the installation, or features require changes).

12.4.8 Questions

Questions can be asked on GitHub discussions.

12.4.9 Issues

For reporting a bug or filing a feature request use GitHub issues.

Do not report security issues on GitHub. Check the support section.

12.5 Package

12.5.1 ska package

12.5.1.1 Subpackages

12.5.1.1.1 ska.contrib package

12.5.1.1.1.1 Subpackages

12.5.1.1.1.2 ska.contrib.django package

12.5.1.1.1.3 Subpackages

12.5.1.1.1.4 ska.contrib.django.ska package

12.5.1.1.1.5 Subpackages

12.5.1.1.1.6 ska.contrib.django.ska.backends package

12.5.1.1.1.7 Submodules

12.5.1.1.1.8 ska.contrib.django.ska.backends.base module

class `ska.contrib.django.ska.backends.base.`**`BaseSkaAuthenticationBackend`**

Bases: `object`

Base authentication backend.

`authenticate`(*request: HttpRequest* | *Request*, ***kwargs*) → User | None

Authenticate.

Parameters

`request` (*`django.http.HttpRequest`*) –

Return django.contrib.auth.models.User

Instance or None on failure.

`get_request_data`(*request: HttpRequest* | *Request*, ***kwargs*) → Dict[str, str]

get_secret_key(*request_data: Dict[str, bytes | str | float | int] | None = None*, *request: HttpRequest | None = None*, ***kwargs*) → str

Get secret key.

Returns

get_settings(*request_data: Dict[str, bytes | str | float | int] | None = None*, *request: HttpRequest | None = None*, ***kwargs*)

Get settings.

Returns

get_user(*user_id: int*) → User

Get user in the `django.contrib.auth.models.User` if exists.

Parameters
user_id (*int*) –

Return django.contrib.auth.models.User

12.5.1.1.1.9 ska.contrib.django.ska.backends.constance_backend module

class ska.contrib.django.ska.backends.constance_backend.**SkaAuthenticationConstanceBackend**

Bases: *BaseSkaAuthenticationBackend*

Authentication backend.

get_secret_key(*request_data: Dict[str, bytes | str | float | int] | None = None*, *request: Request | HttpRequest | None = None*, ***kwargs*) → str

Get secret key.

Returns

get_settings(*request_data: Dict[str, bytes | str | float | int] | None = None*, *request: Request | HttpRequest | None = None*, ***kwargs*) → Dict[str, Dict[str, str]]

Get settings.

Returns

12.5.1.1.1.10 ska.contrib.django.ska.backends.default_backends module

class ska.contrib.django.ska.backends.default_backends.**SkaAuthenticationBackend**

Bases: *BaseSkaAuthenticationBackend*

Authentication backend.

get_secret_key(*request_data: Dict[str, bytes | str | float | int] | None = None*, *request: Request | HttpRequest | None = None*, ***kwargs*) → None

Get secret key.

Returns

get_settings(*request_data: Dict[str, bytes | str | float | int] | None = None*, *request: Request | HttpRequest | None = None*, ***kwargs*) → Dict[Any, Any]

Get settings.

Returns

12.5.1.1.1.11 Module contents

class `ska.contrib.django.ska.backends.`**`BaseSkaAuthenticationBackend`**

Bases: `object`

Base authentication backend.

`authenticate`(*request: HttpRequest* | *Request*, ***kwargs*) → User | None

Authenticate.

Parameters
`request` (*`django.http.HttpRequest`*) –

Return django.contrib.auth.models.User
Instance or None on failure.

`get_request_data`(*request: HttpRequest* | *Request*, ***kwargs*) → Dict[str, str]

`get_secret_key`(*request_data: Dict[str, bytes* | *str* | *float* | *int]* | *None = None*, *request: HttpRequest* | *None = None*, ***kwargs*) → str

Get secret key.

Returns

`get_settings`(*request_data: Dict[str, bytes* | *str* | *float* | *int]* | *None = None*, *request: HttpRequest* | *None = None*, ***kwargs*)

Get settings.

Returns

`get_user`(*user_id: int*) → User

Get user in the `django.contrib.auth.models.User` if exists.

Parameters
`user_id` (*`int`*) –

Return django.contrib.auth.models.User

class `ska.contrib.django.ska.backends.`**`SkaAuthenticationBackend`**

Bases: *`BaseSkaAuthenticationBackend`*

Authentication backend.

`get_secret_key`(*request_data: Dict[str, bytes* | *str* | *float* | *int]* | *None = None*, *request: Request* | *HttpRequest* | *None = None*, ***kwargs*) → None

Get secret key.

Returns

`get_settings`(*request_data: Dict[str, bytes* | *str* | *float* | *int]* | *None = None*, *request: Request* | *HttpRequest* | *None = None*, ***kwargs*) → Dict[Any, Any]

Get settings.

Returns

12.5.1.1.1.12 ska.contrib.django.ska.integration package

12.5.1.1.1.13 Subpackages

12.5.1.1.1.14 ska.contrib.django.ska.integration.drf package

12.5.1.1.1.15 Subpackages

12.5.1.1.1.16 ska.contrib.django.ska.integration.drf.permissions package

12.5.1.1.1.17 Submodules

12.5.1.1.1.18 ska.contrib.django.ska.integration.drf.permissions.base module

class
`ska.contrib.django.ska.integration.drf.permissions.base.`**`AbstractSignedRequestRequired`**

Bases: `BasePermission`

Signed request required permission.

`get_request_data`(*request: Request*, *view: GenericViewSet*, *obj: Model* | *None = None*) → Dict[str, bytes | str | float | int]

`get_secret_key`(*request_data: Dict[str, bytes* | *str* | *float* | *int]*, *request: Request* | *None = None*, *view: GenericViewSet* | *None = None*, *obj: Model* | *None = None*)

Get secret key.

Parameters

- **`request_data`** –
- **`request`** –
- **`view`** –
- **`obj`** –

Returns

`get_settings`(*request_data: Dict[str, bytes* | *str* | *float* | *int]*, *request: Request* | *None = None*, *view: GenericViewSet* | *None = None*, *obj: Model* | *None = None*) → Dict[str, str]

Get settings.

Returns

`has_object_permission`(*request: Request*, *view: GenericViewSet*, *obj: Model*) → bool

Return *True* if permission is granted, *False* otherwise.

`has_permission`(*request: Request*, *view: GenericViewSet*) → bool

Return *True* if permission is granted, *False* otherwise.

`validate_signed_request`(*request: Request*, *view: GenericViewSet*, *obj: Model* | *None = None*) → bool

Validate signed request.

Parameters

- **`request`** –

- **view** –
- **obj** –

Returns

class ska.contrib.django.ska.integration.drf.permissions.base.**BaseProviderSignedRequestRequired**

Bases: *AbstractSignedRequestRequired*

Provider signed request required permission.

get_secret_key(*request_data: Dict[str, str]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → str | None

Get secret key.

Parameters

- **request_data** –
- **request** –
- **view** –
- **obj** –

Returns

class ska.contrib.django.ska.integration.drf.permissions.base.**BaseSignedRequestRequired**

Bases: *AbstractSignedRequestRequired*

Signed request required permission.

get_secret_key(*request_data: Dict[str, str]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → str

Get secret key.

Parameters

- **request_data** –
- **request** –
- **view** –
- **obj** –

Returns

12.5.1.1.1.19 ska.contrib.django.ska.integration.drf.permissions.constance_permissions module

class ska.contrib.django.ska.integration.drf.permissions.constance_permissions.**ConstanceProviderSignedRequestRequired**

Bases: *BaseProviderSignedRequestRequired*

Provider signed request required permission.

get_settings(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → Dict[str, Dict[str, str]]

Get settings.

Returns

class `ska.contrib.django.ska.integration.drf.permissions.constance_permissions.`**`ConstanceSignedRequestRequired`**

Bases: *`BaseSignedRequestRequired`*

Signed request required permission.

`get_settings`(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → Dict[str, str]

Get settings.

Returns

12.5.1.1.1.20 ska.contrib.django.ska.integration.drf.permissions.default_permissions module

class `ska.contrib.django.ska.integration.drf.permissions.default_permissions.`**`ProviderSignedRequestRequired`**

Bases: *`BaseProviderSignedRequestRequired`*

Provider signed request required permission.

`get_settings`(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → Dict[str, Dict[str, str]]

Get settings.

Returns

class `ska.contrib.django.ska.integration.drf.permissions.default_permissions.`**`SignedRequestRequired`**

Bases: *`BaseSignedRequestRequired`*

Signed request required permission.

`get_settings`(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → Dict[str, str]

Get settings.

Returns

12.5.1.1.1.21 Module contents

class `ska.contrib.django.ska.integration.drf.permissions.`**`AbstractSignedRequestRequired`**

Bases: `BasePermission`

Signed request required permission.

`get_request_data`(*request: Request*, *view: GenericViewSet*, *obj: Model | None = None*) → Dict[str, bytes | str | float | int]

`get_secret_key`(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*)

Get secret key.

Parameters

- **`request_data`** –
- **`request`** –

- **view** –
- **obj** –

Returns

get_settings(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → Dict[str, str]

Get settings.

Returns

has_object_permission(*request: Request*, *view: GenericViewSet*, *obj: Model*) → bool

Return *True* if permission is granted, *False* otherwise.

has_permission(*request: Request*, *view: GenericViewSet*) → bool

Return *True* if permission is granted, *False* otherwise.

validate_signed_request(*request: Request*, *view: GenericViewSet*, *obj: Model | None = None*) → bool

Validate signed request.

Parameters

- **request** –
- **view** –
- **obj** –

Returns

class ska.contrib.django.ska.integration.drf.permissions.**BaseProviderSignedRequestRequired**

Bases: *AbstractSignedRequestRequired*

Provider signed request required permission.

get_secret_key(*request_data: Dict[str, str]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → str | None

Get secret key.

Parameters

- **request_data** –
- **request** –
- **view** –
- **obj** –

Returns

class ska.contrib.django.ska.integration.drf.permissions.**BaseSignedRequestRequired**

Bases: *AbstractSignedRequestRequired*

Signed request required permission.

get_secret_key(*request_data: Dict[str, str]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → str

Get secret key.

Parameters

- **request_data** –

- **request** –
- **view** –
- **obj** –

Returns

class ska.contrib.django.ska.integration.drf.permissions.**ProviderSignedRequestRequired**

Bases: *BaseProviderSignedRequestRequired*

Provider signed request required permission.

get_settings(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → Dict[str, Dict[str, str]]

Get settings.

Returns

class ska.contrib.django.ska.integration.drf.permissions.**SignedRequestRequired**

Bases: *BaseSignedRequestRequired*

Signed request required permission.

get_settings(*request_data: Dict[str, bytes | str | float | int]*, *request: Request | None = None*, *view: GenericViewSet | None = None*, *obj: Model | None = None*) → Dict[str, str]

Get settings.

Returns

12.5.1.1.1.22 ska.contrib.django.ska.integration.drf.urls package

12.5.1.1.1.23 Submodules

12.5.1.1.1.24 ska.contrib.django.ska.integration.drf.urls.jwt_token module

12.5.1.1.1.25 Module contents

12.5.1.1.1.26 ska.contrib.django.ska.integration.drf.views package

12.5.1.1.1.27 Submodules

12.5.1.1.1.28 ska.contrib.django.ska.integration.drf.views.jwt_token module

class ska.contrib.django.ska.integration.drf.views.jwt_token.**ObtainJSONWebTokenView**(***kwargs*)

Bases: `APIView`

Obtain a JSON web token.

get(*request: Request*, *format: str | None = None*) → Response

12.5.1.1.1.29 Module contents

12.5.1.1.1.30 Module contents

12.5.1.1.1.31 Module contents

12.5.1.1.1.32 ska.contrib.django.ska.management package

12.5.1.1.1.33 Subpackages

12.5.1.1.1.34 ska.contrib.django.ska.management.commands package

12.5.1.1.1.35 Submodules

12.5.1.1.1.36 ska.contrib.django.ska.management.commands.ska_purge_stored_signature_data module

class `ska.contrib.django.ska.management.commands.ska_purge_stored_signature_data.`**`Command`**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Bases: `BaseCommand`

`handle`(**args*, ***options*)

Purges old signature data (valid_until < now).

12.5.1.1.1.37 Module contents

12.5.1.1.1.38 Module contents

12.5.1.1.1.39 ska.contrib.django.ska.migrations package

12.5.1.1.1.40 Submodules

12.5.1.1.1.41 ska.contrib.django.ska.migrations.0001_initial module

class `ska.contrib.django.ska.migrations.0001_initial.`**`Migration`**(*name*, *app_label*)

Bases: `Migration`

`dependencies = []`

`initial = True`

`operations = [<CreateModel name='Signature', fields=[('id', <django.db.models.fields.AutoField>), ('signature', <django.db.models.fields.CharField>), ('auth_user', <django.db.models.fields.CharField>), ('valid_until', <django.db.models.fields.DateTimeField>), ('created', <django.db.models.fields.DateTimeField>)], options={'verbose_name': 'Token', 'verbose_name_plural': 'Tokens'}>, <AlterUniqueTogether name='signature', unique_together={('signature', 'auth_user', 'valid_until')}>]`

12.5.1.1.1.42 Module contents

12.5.1.1.1.43 ska.contrib.django.ska.templatetags package

12.5.1.1.1.44 Submodules

12.5.1.1.1.45 ska.contrib.django.ska.templatetags.ska_tags module

`ska.contrib.django.ska.templatetags.ska_tags.`**`provider_sign_url`**`(`*context: ~django.template.context.RequestContext, provider: str, url: str = '', auth_user: str | None = None, valid_until: float | str | None = None, lifetime: int = 600, suffix: str = '?', signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra: ~typing.Dict[str, bytes | str | float | int] | None = None, extra_param: str = 'extra', signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>, fail_silently: bool = True*`)` → str | None

Sign URL.

`ska.contrib.django.ska.templatetags.ska_tags.`**`sign_url`**`(`*context: ~django.template.context.RequestContext, url: str = '', auth_user: str | None = None, secret_key: str = 'secret-key', valid_until: float | str | None = None, lifetime: int = 600, suffix: str = '?', signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra: ~typing.Dict[str, bytes | str | float | int] | None = None, extra_param: str = 'extra', signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>*`)` → str

Sign URL.

12.5.1.1.1.46 Module contents

12.5.1.1.1.47 ska.contrib.django.ska.tests package

12.5.1.1.1.48 Submodules

12.5.1.1.1.49 ska.contrib.django.ska.tests.helpers module

`ska.contrib.django.ska.tests.helpers.`**`PROJECT_DIR`**`(`*base*`)`

Project dir.

`ska.contrib.django.ska.tests.helpers.`**`change_date`**`()`

Change date.

`ska.contrib.django.ska.tests.helpers.`**`create_admin_user`**`()`

Create a user for testing the dashboard.

TODO: At the moment an admin account is being tested. Automated tests with diverse accounts are to be implemented.

`ska.contrib.django.ska.tests.helpers.`**`generate_data`**`(`*num_items=5*`)`

Generate data.

`ska.contrib.django.ska.tests.helpers.`**`log_info`**`(`*func*`)`

Logs some useful info.

`ska.contrib.django.ska.tests.helpers.`**`project_dir`**`(`*base*`)`

Project dir.

12.5.1.1.1.50 ska.contrib.django.ska.tests.test_constance_authentication_backend_ module

12.5.1.1.1.51 ska.contrib.django.ska.tests.test_decorators module

`class` `ska.contrib.django.ska.tests.test_decorators.`**`SkaDecoratorsTest`**`(`*methodName='runTest'*`)`

Bases: `TransactionTestCase`

Testing model- and view- decorators.

`pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=(), kwargs={})]`

`setUp``()`

Hook method for setting up the test fixture before exercising it.

`test_01_model_decorator``(`**args*, ***kwargs*`)`

Inner.

`test_02_view_decorator_with_signed_url``(`**args*, ***kwargs*`)`

Inner.

`test_03_view_decorator_with_unsigned_url``(`**args*, ***kwargs*`)`

Inner.

test_04_class_based_view_decorator_with_signed_url(**args*, ***kwargs*)
Inner.

test_05_class_based_view_decorator_with_unsigned_url(**args*, ***kwargs*)
Inner.

12.5.1.1.1.52 ska.contrib.django.ska.tests.test_default_authentication_backend module

class ska.contrib.django.ska.tests.test_default_authentication_backend.**SkaAuthenticationBackendTest**(*meth*

Bases: TransactionTestCase

Tests for auth backend.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=(), kwargs={})]

setUp()
Hook method for setting up the test fixture before exercising it.

test_01_login(**args*, ***kwargs*)
Inner.

test_02_provider_login(**args*, ***kwargs*)
Inner.

test_03_login_fail_wrong_secret_key(**args*, ***kwargs*)
Inner.

test_04_provider_login_fail_wrong_secret_key(**args*, ***kwargs*)
Inner.

test_05_provider_login_fail_wrong_provider(**args*, ***kwargs*)
Inner.

test_06_purge_stored_signatures_data(**args*, ***kwargs*)
Inner.

test_07_provider_login_forbidden_email(**args*, ***kwargs*)
Inner.

test_08_provider_login_forbidden_username(**args*, ***kwargs*)
Inner.

12.5.1.1.1.53 ska.contrib.django.ska.tests.test_drf_integration_permissions module

Testing Django REST Framework permissions for ska.

class ska.contrib.django.ska.tests.test_drf_integration_permissions.**DRFIntegrationPermissionsConstanceT**

Bases: BaseDRFIntegrationPermissionsTestCase

Django REST framework integration permissions constance test case.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=(), kwargs={})]

`test_permissions_detail_request_not_signed_fail()`
Fail test permissions detail request not signed.

Returns

`test_permissions_detail_request_signed()`
Test permissions signed detail request.

Returns

`test_permissions_detail_request_signed_wrong_secret_key_fail()`
Test permissions signed detail request wrong secret key.

Returns

`test_permissions_list_request_not_signed_fail()`
Fail test permissions list request not signed.

Returns

`test_permissions_list_request_signed()`
Test permissions signed list request.

Returns

`test_permissions_list_request_signed_wrong_secret_key_fail()`
Test permissions signed list request wrong secret key.

Returns

`test_permissions_provider_detail_request_not_signed_fail()`
Fail test permissions provider detail request not signed.

Returns

`test_permissions_provider_list_request_not_signed_fail()`
Fail test permissions provider list request not signed.

Returns

`test_provider_permissions_detail_request_signed()`
Test permissions signed provider detail request.

Returns

`test_provider_permissions_detail_request_signed_wrong_secret_key_fail()`
Test permissions signed provider detail request wrong secret key.

Returns

`test_provider_permissions_list_request_signed()`
Test permissions signed provider list request.

Returns

`test_provider_permissions_list_request_signed_wrong_secret_key_fail()`
Test permissions signed provider list request wrong secret key.

Returns

class ska.contrib.django.ska.tests.test_drf_integration_permissions.**DRFIntegrationPermissionsTestCase**(*m*

Bases: BaseDRFIntegrationPermissionsTestCase

Django REST framework integration permissions test case.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=(), kwargs={})]

test_permissions_detail_request_not_signed_fail()

Fail test permissions detail request not signed.

Returns

test_permissions_detail_request_signed()

Test permissions signed detail request.

Returns

test_permissions_detail_request_signed_wrong_secret_key_fail()

Test permissions signed detail request wrong secret key.

Returns

test_permissions_list_request_not_signed_fail()

Fail test permissions list request not signed.

Returns

test_permissions_list_request_signed()

Test permissions signed list request.

Returns

test_permissions_list_request_signed_wrong_secret_key_fail()

Test permissions signed list request wrong secret key.

Returns

test_permissions_provider_detail_request_not_signed_fail()

Fail test permissions provider detail request not signed.

Returns

test_permissions_provider_list_request_not_signed_fail()

Fail test permissions provider list request not signed.

Returns

test_provider_permissions_detail_request_signed()

Test permissions signed provider detail request.

Returns

test_provider_permissions_detail_request_signed_wrong_secret_key_fail()

Test permissions signed provider detail request wrong secret key.

Returns

test_provider_permissions_list_request_signed()

Test permissions signed provider list request.

Returns

test_provider_permissions_list_request_signed_wrong_secret_key_fail()
Test permissions signed provider list request wrong secret key.

Returns

12.5.1.1.1.54 ska.contrib.django.ska.tests.test_drf_integration_view_jwt_token module

Testing Django REST Framework JWT token view for ska.

class `ska.contrib.django.ska.tests.test_drf_integration_view_jwt_token.`**DRFIntegrationViewJwtTokenConsta**

Bases: `BaseDRFIntegrationViewJwtTokenTestCase`

Django REST framework integration view JWT token constance test case.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=(), kwargs={})]

test_obtain_jwt_token_provider_request_signed()
Test provider obtain JWT token signed request.

Returns

test_obtain_jwt_token_provider_request_signed_wrong_secret_key_fail()
Test provider obtain JWT token signed request wrong secret key.

Returns

test_obtain_jwt_token_request_not_signed_fail()
Fail test permissions provider list request not signed.

Returns

test_obtain_jwt_token_request_signed()
Test obtain JWT token signed request.

Returns

test_obtain_jwt_token_request_signed_wrong_secret_key_fail()
Test obtain JWT token signed request wrong secret key.

Returns

class `ska.contrib.django.ska.tests.test_drf_integration_view_jwt_token.`**DRFIntegrationViewJwtTokenTestCa**

Bases: `BaseDRFIntegrationViewJwtTokenTestCase`

Django REST framework integration view JWT token test case.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=(), kwargs={})]

test_obtain_jwt_token_provider_request_signed()
Test provider obtain JWT token signed request.

Returns

test_obtain_jwt_token_provider_request_signed_wrong_secret_key_fail()
Test provider obtain JWT token signed request wrong secret key.

Returns

test_obtain_jwt_token_request_not_signed_fail()

Fail test permissions provider list request not signed.

Returns

test_obtain_jwt_token_request_signed()

Test obtain JWT token signed request.

Returns

test_obtain_jwt_token_request_signed_wrong_secret_key_fail()

Test obtain JWT token signed request wrong secret key.

Returns

12.5.1.1.1.55 Module contents

12.5.1.1.1.56 ska.contrib.django.ska.urls package

12.5.1.1.1.57 Submodules

12.5.1.1.1.58 ska.contrib.django.ska.urls.constance_urls module

12.5.1.1.1.59 ska.contrib.django.ska.urls.default_urls module

12.5.1.1.1.60 Module contents

12.5.1.1.1.61 ska.contrib.django.ska.views package

12.5.1.1.1.62 Submodules

12.5.1.1.1.63 ska.contrib.django.ska.views.constance_views module

ska.contrib.django.ska.views.constance_views.**constance_login**(*request*)

Login.

Authenticate with *ska* token into Django.

Parameters

request (*django.http.HttpRequest*) –

Return django.http.HttpResponse

12.5.1.1.1.64 ska.contrib.django.ska.views.default_views module

`ska.contrib.django.ska.views.default_views.`**`login`**(*request*)

Login.

Authenticate with *ska* token into Django.

Parameters
`request` (*`django.http.HttpRequest`*) –

Return django.http.HttpResponse

12.5.1.1.1.65 Module contents

`ska.contrib.django.ska.views.`**`constance_login`**(*request*)

Login.

Authenticate with *ska* token into Django.

Parameters
`request` (*`django.http.HttpRequest`*) –

Return django.http.HttpResponse

`ska.contrib.django.ska.views.`**`login`**(*request*)

Login.

Authenticate with *ska* token into Django.

Parameters
`request` (*`django.http.HttpRequest`*) –

Return django.http.HttpResponse

12.5.1.1.1.66 Submodules

12.5.1.1.1.67 ska.contrib.django.ska.admin module

`class` `ska.contrib.django.ska.admin.`**`SignatureAdmin`**(*model*, *admin_site*)

Bases: `ModelAdmin`

Signature admin.

`class Meta`

Bases: `object`

Meta class.

`app_label = 'Signature'`

`fieldsets = ((None, {'fields': ('signature', 'auth_user', 'valid_until')}), ('Additional', {'classes': ('collapse',), 'fields': ('created',)}))`

`list_display = ('signature', 'auth_user', 'valid_until', 'created')`

`list_filter = ('auth_user',)`

property media

readonly_fields = ('created',)

12.5.1.1.1.68 ska.contrib.django.ska.apps module

class `ska.contrib.django.ska.apps.`**`Config`**(*app_name*, *app_module*)

Bases: `AppConfig`

Config.

label = 'ska'

name = 'ska.contrib.django.ska'

12.5.1.1.1.69 ska.contrib.django.ska.conf module

`ska.contrib.django.ska.conf.`**`get_setting`**(*setting*, *override=None*)

Get a setting from *ska.contrib.django.ska* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

12.5.1.1.1.70 ska.contrib.django.ska.decorators module

- `validate_signed_request`: Function decorator. Validate request signature. Applies appropriate validation mechanism to the request data. Assumes `SKA_SECRET_KEY` to be in `settings` module.

  Arguments to be used with *ska.validate_signed_request_data* shortcut function.

  param str secret_key
  The shared secret key.

  param str signature_param
  Name of the (for example GET or POST) param name which holds the `signature` value.

  param str auth_user_param
  Name of the (for example GET or POST) param name which holds the `auth_user` value.

  param str valid_until_param
  Name of the (foe example GET or POST) param name which holds the `valid_until` value.

- `sign_url`: Method decorator (to be used in models). Signs the URL.

  Arguments to be used with *ska.sign_url* shortcut function.

  param str auth_user
  Username of the user making the request.

  param str secret_key
  The shared secret key.

  param float|str valid_until
  Unix timestamp. If not given, generated automatically (now + lifetime).

  param int lifetime
  Signature lifetime in seconds.

param str suffix
Suffix to add after the `endpoint_url` and before the appended signature params.

param str signature_param
Name of the GET param name which would hold the generated signature value.

param str auth_user_param
Name of the GET param name which would hold the `auth_user` value.

param str valid_until_param
Name of the GET param name which would hold the `valid_until` value.

`class ska.contrib.django.ska.decorators.BaseValidateSignedRequest`(*secret_key: str = 'secret-key'*, *signature_param: str = 'signature'*, *auth_user_param: str = 'auth_user'*, *valid_until_param: str = 'valid_until'*, *extra_param: str = 'extra'*)

Bases: `object`

BaseValidateSignedRequest.

`get_request_data`(*request: HttpRequest*, **args*, ***kwargs*) → Dict[str, str]

`class ska.contrib.django.ska.decorators.MethodValidateSignedRequest`(*secret_key: str = 'secret-key'*, *signature_param: str = 'signature'*, *auth_user_param: str = 'auth_user'*, *valid_until_param: str = 'valid_until'*, *extra_param: str = 'extra'*)

Bases: *BaseValidateSignedRequest*

MethodValidateSignedRequest.

Method decorator. Validate request signature. Applies appropriate validation mechanism to the request data. Assumes `SKA_SECRET_KEY` to be in `settings` module.

Arguments to be used with *ska.validate_signed_request_data* shortcut function.

Attribute str secret_key
The shared secret key.

Attribute str signature_param
Name of the (for example GET or POST) param name which holds the `signature` value.

Attribute str auth_user_param
Name of the (for example GET or POST) param name which holds the `auth_user` value.

Attribute str valid_until_param
Name of the (foe example GET or POST) param name which holds the `valid_until` value.

Attribute str extra_param
Name of the (foe example GET or POST) param name which holds the `extra` value.

Example

```
>>> from ska.contrib.django.ska.decorators import m_validate_signed_request
>>>
>>> class FooDetailView(View):
>>>     @validate_signed_request()
>>>     def get(self, request, slug, template_name='foo/detail.html'):
>>>         # Your code
```

class `ska.contrib.django.ska.decorators.`**`SignAbsoluteURL`**(*auth_user: str = 'ska-auth-user', secret_key: str = 'secret-key', valid_until: float | str | None = None, lifetime: int = 600, suffix: str = '?', signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra: Dict[str, bytes | str | float | int] | None = None, extra_param: str = 'extra'*)

Bases: `object`

SignAbsoluteURL.

Method decorator (to be used in models). Signs the URL.

Arguments to be used with *ska.sign_url* shortcut function.

Attribute str auth_user
Username of the user making the request.

Attribute str secret_key
The shared secret key.

Attribute float | str valid_until
Unix timestamp. If not given, generated automatically (now + lifetime).

Attribute int lifetime
Signature lifetime in seconds.

Attribute str suffix
Suffix to add after the `endpoint_url` and before the appended signature params.

Attribute str signature_param
Name of the GET param name which would hold the generated signature value.

Attribute str auth_user_param
Name of the GET param name which would hold the `auth_user` value.

Attribute str valid_until_param
Name of the GET param name which would hold the `valid_until` value.

Attribute dict extra
Dict of extra params to append to signed URL.

Attribute str extra_param
Name of the GET param name which would hold the `extra` value.

Example

```
>>> from ska.contrib.django.ska.decorators import sign_url
>>>
>>> class FooItem(models.Model):
```

(continues on next page)

(continued from previous page)

```
>>>     title = models.CharField(_("Title"), max_length=100)
>>>     slug = models.SlugField(unique=True, verbose_name=_("Slug"))
>>>     body = models.TextField(_("Body"))
>>>
>>>     @sign_url()
>>>     def get_signed_absolute_url(self):
>>>         return reverse('foo.detail', kwargs={'slug': self.slug})
```

class ska.contrib.django.ska.decorators.**ValidateSignedRequest**(*secret_key: str = 'secret-key'*, *signature_param: str = 'signature'*, *auth_user_param: str = 'auth_user'*, *valid_until_param: str = 'valid_until'*, *extra_param: str = 'extra'*)

Bases: *BaseValidateSignedRequest*

ValidateSignedRequest.

Function decorator. Validate request signature. Applies appropriate validation mechanism to the request data. Assumes `SKA_SECRET_KEY` to be in `settings` module.

Arguments to be used with *ska.validate_signed_request_data* shortcut function.

Attribute str secret_key
The shared secret key.

Attribute str signature_param
Name of the (for example GET or POST) param name which holds the `signature` value.

Attribute str auth_user_param
Name of the (for example GET or POST) param name which holds the `auth_user` value.

Attribute str valid_until_param
Name of the (foe example GET or POST) param name which holds the `valid_until` value.

Attribute str extra_param
Name of the (foe example GET or POST) param name which holds the `extra` value.

Example

```
>>> from ska.contrib.django.ska.decorators import validate_signed_request
>>>
>>> @validate_signed_request()
>>> def detail(request, slug, template_name='foo/detail.html'):
>>>     # Your code
```

ska.contrib.django.ska.decorators.**m_validate_signed_request**
alias of *MethodValidateSignedRequest*

ska.contrib.django.ska.decorators.**sign_url**
alias of *SignAbsoluteURL*

ska.contrib.django.ska.decorators.**validate_signed_request**
alias of *ValidateSignedRequest*

12.5.1.1.1.71 ska.contrib.django.ska.defaults module

- *UNAUTHORISED_REQUEST_ERROR_MESSAGE* (str): Plain text error message. Defaults to "Unauthorised request. {0}".
- *UNAUTHORISED_REQUEST_ERROR_TEMPLATE* (str): Path to 401 template that should be rendered in case of 401 responses. Defaults to empty string (not provided).
- *AUTH_USER* (str): Default `auth_user` for `ska.sign_url` function. Defaults to "ska-auth-user".
- *USER_GET_CALLBACK* (str): User get callback (when user is fetched in auth backend).
- *USER_VALIDATE_CALLBACK* (str): User validate callback (fired before user is created; created to allow custom logic to the user authentication before user object is even created).
- *USER_CREATE_CALLBACK* (str): User create callback (when user is created in auth backend).
- *USER_INFO_CALLBACK* (str): User info callback.
- *REDIRECT_AFTER_LOGIN* (str): Redirect after login.
- *DB_STORE_SIGNATURES* (bool): If set to True, signatures are stored in the database.
- *DB_PERFORM_SIGNATURE_CHECK* (bool): If set to True, an extra check is fired on whether the token has already been used or not.
- *PROVIDERS* (dict): A dictionary where key is the provider UID and the key is another dictionary holding the following provider specific keys: 'SECRET_KEY', 'USER_GET_CALLBACK', 'USER_CREATE_CALLBACK', 'USER_INFO_CALLBACK', 'REDIRECT_AFTER_LOGIN'. Note, that the 'SECRET_KEY' is a required key. The rest are optional, and if given, override respectively the values of `ska.contrib.django.ska.settings`.

12.5.1.1.1.72 ska.contrib.django.ska.http module

class `ska.contrib.django.ska.http.`**`HttpResponseUnauthorized`**(*content=b''*, **args*, ***kwargs*)

Bases: `HttpResponseForbidden`

HttpResponseUnauthorized.

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#4xx_Client_Error

`status_code = 401`

12.5.1.1.1.73 ska.contrib.django.ska.models module

class `ska.contrib.django.ska.models.`**`Signature`**(**args*, ***kwargs*)

Bases: `Model`

Signature.

Properties

- *signature* (str): Signature generated.
- *auth_user* (str): Auth user.
- *valid_until* (datetime.datetime): Valid until.
- *created* (datetime.datetime): Time added.

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

auth_user

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

created

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_created(**, field=<django.db.models.fields.DateTimeField: created>, is_next=True, **kwargs*)

get_next_by_valid_until(**, field=<django.db.models.fields.DateTimeField: valid_until>, is_next=True, **kwargs*)

get_previous_by_created(**, field=<django.db.models.fields.DateTimeField: created>, is_next=False, **kwargs*)

get_previous_by_valid_until(**, field=<django.db.models.fields.DateTimeField: valid_until>, is_next=False, **kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

signature

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

valid_until

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

12.5.1.1.1.74 ska.contrib.django.ska.settings module

- *UNAUTHORISED_REQUEST_ERROR_MESSAGE* (str): Plain text error message. Defaults to “Unauthorised request. {0}”.
- *UNAUTHORISED_REQUEST_ERROR_TEMPLATE* (str): Path to 401 template that should be rendered in case of 401 responses. Defaults to empty string (not provided).
- *AUTH_USER* (str): Default `auth_user` for `ska.sign_url` function. Defaults to “ska-auth-user”.
- *SECRET_KEY* (str): The shared secret key. Should be defined in *settings* module as `SKA_SECRET_KEY`.
- *USER_GET_CALLBACK* (str): User get callback (when user is fetched in auth backend).
- *USER_VALIDATE_CALLBACK* (str): User validate callback (fired before user is created; created to allow custom logic to the user authentication before user object is even created).
- *USER_CREATE_CALLBACK* (str): User create callback (when user is created in auth backend).

- *USER_INFO_CALLBACK* (str): User info callback.
- *REDIRECT_AFTER_LOGIN* (str): Redirect after login.
- *DB_STORE_SIGNATURES* (bool): If set to True, signatures are stored in the database.
- *DB_PERFORM_SIGNATURE_CHECK* (bool): If set to True, an extra check is fired on whether the token has already been used or not.
- *PROVIDERS* (dict): A dictionary where key is the provider UID and the key is another dictionary holding the following provider specific keys: 'SECRET_KEY', 'USER_GET_CALLBACK', 'USER_CREATE_CALLBACK', 'USER_INFO_CALLBACK', 'REDIRECT_AFTER_LOGIN'. Note, that the 'SECRET_KEY' is a required key. The rest are optional, and if given, override respectively the values of `ska.contrib.django.ska.settings`.

12.5.1.1.1.75 ska.contrib.django.ska.utils module

`ska.contrib.django.ska.utils.`**`get_provider_data`**`(`*data: Dict[str, bytes | str | float | int], settings: Dict[str, Dict[str, str]] | None = None*`)` → Dict[str, str] | None

Obtain the secret key from request data given.

This happens by looking up the secret key by *provider* param from the request data in the dictionary of `PROVIDERS` defined in settings module. If not found, fall back to the `default` value given, which is by default the globally set secret key.

Parameters

- **`data`** (*`dict`*) –
- **`settings`** (*`dict`*) – Settings dict.

`ska.contrib.django.ska.utils.`**`get_secret_key`**`(`*data: Dict[str, bytes | str | float | int] | None, default: str = 'secret-key'*`)` → str

Obtain the secret key from request data given.

This happens by looking up the secret key by *provider* param from the request data in the dictionary of `PROVIDERS` defined in settings module. If not found, fall back to the `default` value given, which is by default the globally set secret key.

Parameters

- **`data`** (*`dict`*) –
- **`default`** (*`string`*) – Secret key value to be used as default. By default, the globally set secret key is used.

`ska.contrib.django.ska.utils.`**`purge_signature_data`**`()` → None

Purge old signature data (valid_until < now).

12.5.1.1.1.76 Module contents

12.5.1.1.1.77 Module contents

12.5.1.1.1.78 Module contents

12.5.1.1.2 ska.signatures package

12.5.1.1.2.1 Submodules

12.5.1.1.2.2 ska.signatures.hmac_md5 module

`class ska.signatures.hmac_md5.`**`HMACMD5Signature`**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *`AbstractSignature`*

HMAC MD5 signature.

`auth_user`

`extra`

`classmethod` **`make_hash`**(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **`auth_user`** –
- **`secret_key`** –
- **`valid_until`** – Unix timestamp, valid until.
- **`extra`** – Additional variables to be added.
- **`value_dumper`** –
- **`quoter`** –

Returns

`signature`

`valid_until`

12.5.1.1.2.3 ska.signatures.hmac_sha1 module

`class ska.signatures.hmac_sha1.`**`HMACSHA1Signature`**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `AbstractSignature`

HMAC SHA-1 signature.

`auth_user`

`extra`

`classmethod` **`make_hash`**(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **`auth_user`** –
- **`secret_key`** –
- **`valid_until`** – Unix timestamp, valid until.
- **`extra`** – Additional variables to be added.
- **`value_dumper`** –
- **`quoter`** –

Returns

`signature`

`valid_until`

12.5.1.1.2.4 ska.signatures.hmac_sha224 module

`class ska.signatures.hmac_sha224.`**`HMACSHA224Signature`**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `AbstractSignature`

HMAC SHA-224 signature.

`auth_user`

`extra`

`classmethod` **`make_hash`**(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

12.5.1.1.2.5 ska.signatures.hmac_sha256 module

class ska.signatures.hmac_sha256.**HMACSHA256Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC SHA-256 signature.

auth_user

extra

classmethod make_hash(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

12.5.1.1.2.6 ska.signatures.hmac_sha384 module

`class ska.signatures.hmac_sha384.`**`HMACSHA384Signature`**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `AbstractSignature`

HMAC SHA-384 signature.

`auth_user`

`extra`

`classmethod` **`make_hash`**(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **`auth_user`** –
- **`secret_key`** –
- **`valid_until`** – Unix timestamp, valid until.
- **`extra`** – Additional variables to be added.
- **`value_dumper`** –
- **`quoter`** –

Returns

`signature`

`valid_until`

12.5.1.1.2.7 ska.signatures.hmac_sha512 module

`class ska.signatures.hmac_sha512.`**`HMACSHA512Signature`**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `AbstractSignature`

HMAC SHA-512 signature.

`auth_user`

`extra`

`classmethod` **`make_hash`**(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

12.5.1.1.2.8 Module contents

class ska.signatures.**HMACMD5Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC MD5 signature.

auth_user

extra

classmethod make_hash(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

class ska.signatures.**HMACSHA1Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC SHA-1 signature.

`auth_user`

`extra`

`classmethod make_hash`(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **`auth_user`** –
- **`secret_key`** –
- **`valid_until`** – Unix timestamp, valid until.
- **`extra`** – Additional variables to be added.
- **`value_dumper`** –
- **`quoter`** –

Returns

`signature`

`valid_until`

`class ska.signatures.HMACSHA224Signature`(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `AbstractSignature`

HMAC SHA-224 signature.

`auth_user`

`extra`

`classmethod make_hash`(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **`auth_user`** –
- **`secret_key`** –
- **`valid_until`** – Unix timestamp, valid until.
- **`extra`** – Additional variables to be added.
- **`value_dumper`** –
- **`quoter`** –

Returns

`signature`

`valid_until`

`class ska.signatures.HMACSHA256Signature`(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `AbstractSignature`

HMAC SHA-256 signature.

`auth_user`

`extra`

`classmethod make_hash`(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

`signature`

`valid_until`

`class ska.signatures.HMACSHA384Signature`(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `AbstractSignature`

HMAC SHA-384 signature.

`auth_user`

`extra`

`classmethod make_hash`(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.

- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

class ska.signatures.**HMACSHA512Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC SHA-512 signature.

auth_user

extra

classmethod make_hash(*auth_user: str*, *secret_key: str*, *valid_until: str | float | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

ska.signatures.**Signature**

alias of *HMACSHA1Signature*

12.5.1.1.3 ska.tests package

12.5.1.1.3.1 Submodules

12.5.1.1.3.2 ska.tests.base module

ska.tests.base.**log_info**(*func*)

Prints some useful info.

`ska.tests.base.`**`parse_url_params`**`(`*url*`)`

Parses URL params.

Parameters
`url` (*`str`*) –

Return dict

`ska.tests.base.`**`timestamp_to_human_readable`**`(`*timestamp*`)`

Convert Unix timestamp to human readable string.

Parameters
`timestamp` –

Return str

12.5.1.1.3.3 ska.tests.test_commands module

`class` `ska.tests.test_commands.`**`GenerateSignedUrlTest`**`(`*methodName='runTest'*`)`

Bases: `TestCase`

Tests of *generate_signed_url* module and *ska-sign-url* script.

`setUp``()`

Set up.

`test_generate_signed_url``()`

Test *generate_signed_url* module.

Returns

12.5.1.1.3.4 ska.tests.test_core module

`class` `ska.tests.test_core.`**`ExtraTest`**`(`*methodName='runTest'*`)`

Bases: `TestCase`

Test for extra data.

`setUp``()`

Set up.

`test_01_sign_url_and_validate_signed_request_data``()`

Tests for `sign_url` and `validate_signed_request_data`.

`test_02_sign_url_validate_signed_req_data_tamper_extra_keys_rm``()`

Fail tests for *sign_url* and *validate_signed_request_data*.

As well as providing the additional data *extra* and data tampering *extra* keys (remove).

`test_03_sign_url_and_validate_signed_req_data_tamper_extra_keys_add``()`

Fail tests for *sign_url* and *validate_signed_request_data*.

As well as providing the additional data `extra` and data tampering *extra* keys (add).

`test_04_sgn_url_vldt_signed_request_data_tamper_extra_keys_add``()`

Tests for *sign_url* and *validate_signed_request_data*.

As well as providing the additional data *extra* and data tampering *extra* keys (add) repeated params.

`class ska.tests.test_core.ShortcutsTest(methodName='runTest')`

Bases: `TestCase`

Tests for shortcut functions.

The following shortcut functions are tested: *sign_url*, *signature_to_dict* and *validate_signed_request_data*.

`setUp()`

Set up.

`test_01_sign_url_and_validate_signed_request_data()`

Tests for `sign_url` & `validate_signed_request_data`.

`test_02_sign_url_and_validate_signed_request_data_fail()`

Fail tests for *sign_url* & *validate_signed_request_data*.

`test_03_signature_to_dict_and_validate_signed_request_data()`

Tests for *signature_to_dict* & *validate_signed_request_data*.

`test_04_sig_to_dict_var_types_and_validate_signed_request_data()`

Tests for *signature_to_dict* with complex data & *validate_signed_request_data*.

`class ska.tests.test_core.SignatureTest(methodName='runTest')`

Bases: `TestCase`

Tests of *ska.Signature* class.

`setUp()`

Set up.

`test_01_signature_test()`

Signature test.

`test_02_signature_test_with_positive_time_lapse()`

Signature test with positive time-lapse.

When signature is made on a host that has a positive (greater) time difference with server. In this particular example, the host time is 5 minutes ahead the server time.

`test_03_signature_test_with_negative_time_lapse()`

Fail test. Signature test with negative time-lapse.

When signature is made on a host that has a negative (less) time difference with server. In this particular example, the host time is 5 minutes behind the server time, which exceeds the signature lifetime.

`test_04_fail_signature_test()`

Fail signature tests.

`test_05_fail_signature_test_validation_result_class()`

Fail signature tests of the *ValidationResult* class.

`class ska.tests.test_core.URLHelperTest(methodName='runTest')`

Bases: `TestCase`

Tests of *ska.URLHelper* class.

`setUp()`

Set up.

test_01_signature_to_url()

Signature test.

test_02_signature_to_url_fail()

Signature test. Fail test.

12.5.1.1.3.5 Module contents

12.5.1.2 Submodules

12.5.1.3 ska.base module

class ska.base.**AbstractSignature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: `object`

Abstract class for signature generation and validation.

Based on symmetric keys.

Parameters

- **signature** –
- **auth_user** –
- **valid_until** –

auth_user

static datetime_to_timestamp(*dtv: datetime*) → str | None

Human readable datetime according to the format specified.

Format is specified in `TIMESTAMP_FORMAT`.

Parameters
dtv –

Returns

static datetime_to_unix_timestamp(*dtv: datetime*) → float | None

Convert `datetime.datetime` to Unix timestamp.

Parameters
dtv –

Returns
Unix timestamp.

extra

classmethod generate_signature(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *lifetime: int = 600*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → *AbstractSignature*

Generates the signature.

If timestamp is given, the signature is created using the given timestamp. Otherwise current time is used.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **lifetime** – Lifetime of the signature in seconds.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

Example

```
>>> sig = Signature.generate_signature('user', 'your-secret-key')
EBS6ipiqRLa6TY5vxIvZU30FpnM=
```

classmethod get_base(*auth_user: str*, *timestamp: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Get base string.

Add something here so that timestamp to signature conversion is not that obvious.

Parameters

- **auth_user** –
- **timestamp** –
- **extra** –
- **value_dumper** –
- **quoter** –

is_expired() → bool

Checks if current signature is expired.

Returns True if signature is expired and False otherwise.

Returns

Example

```
>>> # Generating the signature
>>> sig = Signature.generate_signature('user', 'your-secret-key')
>>> sig.is_expired()
False
```

classmethod make_hash(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

`static make_secret_key`(*secret_key: str*) → bytes

The secret key how its' supposed to be used in generate signature.

Parameters
secret_key –

Returns

`signature`

`classmethod timestamp_to_date`(*timestamp: float | str*, *fail_silently: bool = True*) → datetime | None

Converts the given timestamp to date.

If `fail_silently` is set to False, raises exceptions if timestamp is not valid timestamp (according to the format we have specified in the `TIMESTAMP_FORMAT`). Mainly used internally.

Parameters

- **timestamp** –
- **fail_silently** –

Returns

`classmethod unix_timestamp_to_date`(*timestamp: float | str*, *fail_silently: bool = True*) → datetime | None

Converts the given Unix timestamp to date. If `fail_silently` is set to False, raises exceptions if timestamp is not valid timestamp.

Parameters

- **timestamp** – UNIX timestamp. Possible to parse to float.
- **fail_silently** –

Returns

`valid_until`

`classmethod validate_signature`(*signature: str | bytes*, *auth_user: str*, *secret_key: str*, *valid_until: str | float*, *extra: Dict[str, bytes | str | float | int] | None = None*, *return_object: bool = False*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → *SignatureValidationResult* | bool

Validates the signature.

Parameters

- **signature** –
- **auth_user** –
- **secret_key** –

- **valid_until** – Unix timestamp.
- **extra** – Extra arguments to be validated.
- **return_object** – If set to True, an instance of `SignatureValidationResult` is returned.
- **value_dumper** –
- **quoter** –

Returns

Example

```
>>> Signature.validate_signature(
>>>     'EBS6ipiqRLa6TY5vxIvZU30FpnM=',
>>>     'user',
>>>     'your-secret-key',
>>>     '1377997396.0'
>>> )
False
```

class ska.base.**SignatureValidationResult**(*result: bool*, *errors: List[ErrorCode | Any] | None = None*)

Bases: `object`

Signature validation result container.

If signature validation result is True, things like this would work:

```
>>> res = SignatureValidationResult(result=True)
>>> print bool(res)
True
>>> res = SignatureValidationResult(
>>>     result=False,
>>>     reason=[error_codes.INVALID_SIGNATURE,]
>>> )
>>> print bool(res)
False
```

property message: str

Human readable message of all errors.

Returns

property reason: map

Reason.

For backwards compatibility. Returns list of text messages.

Returns

12.5.1.4 ska.defaults module

Application defaults.

- *SIGNATURE_LIFETIME* (int): Signature lifetime in seconds. Default value is 600 (seconds).
- *DEFAULT_SIGNATURE_PARAM* (str): Default name of the REQUEST param holding the generated signature value. Default value is *signature*.
- *DEFAULT_AUTH_USER_PARAM* (str): Default name of the REQUEST param holding the `auth_user` value. Default value is *auth_user*.
- *DEFAULT_VALID_UNTIL_PARAM* (str): Default name of the REQUEST param holding the `valid_until` value. Default value is *valid_until*.
- *DEFAULT_TIME_ZONE_PARAM* (str): Default name of the REQUEST param holding the `time_zone` value. Default value is *time_zone*.
- *DEFAULT_EXTRA_PARAM* (str): Default name of the REQUEST param holding the `extra` value. Default value is *extra*.
- *DEFAULT_PROVIDER_PARAM* (str): Default name of the REQUEST param holding the `provider` value. Default value is *provider*.
- *DEFAULT_URL_SUFFIX* (str): Suffix to add after the `endpoint_url` and before the appended signature params.
- *DEFAULT_RESERVED_PARAMS* (list): List of GET params reserved by default. Users should not be allowed to use them.

12.5.1.5 ska.error_codes module

`class ska.error_codes.`**`ErrorCode`**(*code: int*, *message: str*)

Bases: `object`

Base error code.

If you have ever used the following code with *validation_result*:

```
>>> human_readable_error = ' '.join(validation_result.reason)
```

. . . change it as follows:

```
>>> human_readable_error = validation_result.message
```

Property int code
Just an integer code.

Property string message
Human readable represantation of the error message.

`code`

`message`

12.5.1.6 ska.exceptions module

exception ska.exceptions.**BaseSkaException**

Bases: `Exception`

Base exception.

exception ska.exceptions.**ImproperlyConfigured**

Bases: *BaseSkaException*

Improperly configured exception.

Raised when developer didn't configure/write the code properly.

exception ska.exceptions.**InvalidData**

Bases: *BaseSkaException*

Invalid data exception.

Raised when invalid data (tampered) is detected.

12.5.1.7 ska.generate_signed_url module

ska.generate_signed_url.**main**()

Prints signed URL to console.

Example
python src/ska/generate_signature.py -u http://example.com -au user -sk test

Example
ska-sign-url -u http://example.com -au user -sk test

12.5.1.8 ska.gettext module

12.5.1.9 ska.helpers module

ska.helpers.**default_quoter**(*value*)

ska.helpers.**default_value_dumper**(*value*)

ska.helpers.**dict_keys**(*data: Dict[str, bytes | str | float | int]*, *return_string: bool = False*) → str | List[str]

Get sorted keys from dictionary given.

If `return_string` argument is set to True, returns keys joined by commas.

Parameters

- **data** –
- **return_string** –

Returns

ska.helpers.**dict_to_ordered_list**(*data: Dict[str, bytes | str | float | int]*) → List[Tuple[str, bytes | str | float | int]]

Get extra as ordered list.

Parameters
data (*dict*) –

Returns

`ska.helpers.`**`extract_signed_data`**(*data: Dict[str, bytes | str | float | int]*, *extra: List[str]*) → Dict[str, bytes | str | float | int]

Filters out non-white-listed items from the `extra` dictionary given.

Parameters

- **`data`** –
- **`extra`** –

Returns

`ska.helpers.`**`get_callback_func`**(*func: str | Callable*, *fail_silently: bool = True*) → Callable | None

Take a string and try to extract a function from it.

Parameters

- **`func`** – If *callable* is given, return as is. If *str* is given, try to extract the function from the string given and return.
- **`fail_silently`** –

Returns

Returns *callable* if what's extracted is callable or None otherwise.

`ska.helpers.`**`javascript_quoter`**(*value*)

`ska.helpers.`**`javascript_value_dumper`**(*value*)

`ska.helpers.`**`make_valid_until`**(*lifetime: int = 600*) → float

Make valid until.

Parameters

`lifetime` –

Returns

`ska.helpers.`**`sorted_urlencode`**(*data: ~typing.Dict[str, bytes | str | float | int], quoted: bool = True, value_dumper: ~typing.Callable | None = <function default_value_dumper>, quoter: ~typing.Callable | None = <function default_quoter>*) → str

Similar to built-in `urlencode`, but always puts data in a sorted constant way that stays the same between various python versions.

Parameters

- **`data`** –
- **`quoted`** –
- **`value_dumper`** –
- **`quoter`** –

Returns

12.5.1.10 ska.shortcuts module

`ska.shortcuts.extract_signed_request_data`(*data: ~typing.Dict[str, bytes | str | float | int], secret_key: str | None = None, signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra_param: str = 'extra', validate: bool = False, fail_silently: bool = False, signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>, value_dumper: ~typing.Callable | None = None, quoter: ~typing.Callable | None = None*) → Dict[str, bytes | str | float | int]

Validate the signed request data.

Parameters

- `data` – Dictionary holding the (HTTP) request (for example GET or POST) data.
- `secret_key` – The shared secret key.
- `signature_param` – Name of the (for example GET or POST) param name which holds the `signature` value.
- `auth_user_param` – Name of the (for example GET or POST) param name which holds the `auth_user` value.
- `valid_until_param` – Name of the (foe example GET or POST) param name which holds the `valid_until` value.
- `extra_param` – Name of the (foe example GET or POST) param name which holds the `extra` value.
- `validate` – If set to True, request data is validated before returning the result.
- `fail_silently` – If set to True, exceptions are omitted.
- `signature_cls` –
- `value_dumper` –
- `quoter` –

Returns

Dictionary with signed request data.

`ska.shortcuts.sign_url`(*auth_user: str, secret_key: str, valid_until: float | str | None = None, lifetime: int = 600, url: str = '', suffix: str = '?', signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra: ~typing.Dict[str, bytes | str | float | int] | None = None, extra_param: str = 'extra', signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>, value_dumper: ~typing.Callable | None = None*) → str

Sign the URL.

Parameters

- `auth_user` – Username of the user making the request.
- `secret_key` – The shared secret key.
- `valid_until` – Unix timestamp. If not given, generated automatically (now + lifetime).
- `lifetime` – Signature lifetime in seconds.

- **`url`** – URL to be signed.
- **`suffix`** – Suffix to add after the `endpoint_url` and before the appended signature params.
- **`signature_param`** – Name of the GET param name which would hold the generated signature value.
- **`auth_user_param`** – Name of the GET param name which would hold the `auth_user` value.
- **`valid_until_param`** – Name of the GET param name which would hold the `valid_until` value.
- **`extra`** – Extra variables to add to the request.
- **`extra_param`** – Name of the GET param name which would hold the `extra_keys` value.
- **`signature_cls`** –
- **`value_dumper`** –

Returns

Example

Required imports.

```
>>> from ska import sign_url
```

Producing a signed URL.

```
>>> signed_url = sign_url(
>>>     auth_user='user', secret_key='your-secret_key', lifetime=120,
>>>     url='http://e.com/api/', signature_param=DEFAULT_SIGNATURE_PARAM,
>>>     auth_user_param=DEFAULT_AUTH_USER_PARAM,
>>>     valid_until_param=DEFAULT_VALID_UNTIL_PARAM,
>>>     extra={
>>>         'provider': 'service1.example.com',
>>>         'email': 'john.doe@mail.example.com'
>>>     },
>>>     extra_param = DEFAULT_EXTRA_PARAM
>>> )
http://e.com/api/?valid_until=1378045287.0&auth_user=user&signature=
YlZpLFsjUKBalL4x5trhkeEgqE8%3D
```

`ska.shortcuts.`**`signature_to_dict`**(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *lifetime: int = 600*, *signature_param: str = 'signature'*, *auth_user_param: str = 'auth_user'*, *valid_until_param: str = 'valid_until'*, *extra: ~typing.Dict[str, str | int] | None = None*, *extra_param: str = 'extra'*, *signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>*, *value_dumper: ~typing.Callable | None = None*, *quoter: ~typing.Callable | None = None*) → Dict[str, bytes | str | float | int]

Return a dictionary containing the signature data params.

Parameters

- **`auth_user`** – Username of the user making the request.
- **`secret_key`** – The shared secret key.
- **`valid_until`** – Unix timestamp. If not given, generated automatically (now + lifetime).

- **`lifetime`** – Signature lifetime in seconds.
- **`signature_param`** – Name of the (for example POST) param name which would hold the generated `signature` value.
- **`auth_user_param`** – Name of the (for example POST) param name which would hold the `auth_user` value.
- **`valid_until_param`** – Name of the (for example POST) param name which would hold the `valid_until` value.
- **`extra`** – Additional arguments for the signature.
- **`extra_param`** – Name of the (for example POST) param name which would hold the `extra` keys value.
- **`signature_cls`** –
- **`value_dumper`** –
- **`quoter`** –

Returns

Example

Required imports.

```
>>> from ska import signature_to_dict
```

Producing a dictionary with signature data.

```
>>> signature_dict = signature_to_dict(
>>>     auth_user='user', secret_key='your-secret_key', lifetime=120,
>>>     signature_param=DEFAULT_SIGNATURE_PARAM,
>>>     auth_user_param=DEFAULT_AUTH_USER_PARAM,
>>>     valid_until_param=DEFAULT_VALID_UNTIL_PARAM
>>> )
{
    'signature': 'YlZpLFsjUKBalL4x5trhkeEgqE8=',
    'auth_user': 'user',
    'valid_until': '1378045287.0'
}
```

`ska.shortcuts.`**`validate_signed_request_data`**(*data: ~typing.Dict[str, bytes | str | float | int], secret_key: str, signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra_param: str = 'extra', signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>, value_dumper: ~typing.Callable | None = None, quoter: ~typing.Callable | None = None*) → *SignatureValidationResult*

Validate the signed request data.

Parameters

- **`data`** – Dictionary holding the (HTTP) request (for example GET or POST) data.
- **`secret_key`** – The shared secret key.

- **signature_param** – Name of the (for example GET or POST) param name which holds the `signature` value.
- **auth_user_param** – Name of the (for example GET or POST) param name which holds the `auth_user` value.
- **valid_until_param** – Name of the (foe example GET or POST) param name which holds the `valid_until` value.
- **extra_param** – Name of the (foe example GET or POST) param name which holds the `extra` keys value.
- **signature_cls** –
- **value_dumper** –
- **quoter** –

Returns

A `ska.SignatureValidationResult` object with the following properties:

- *result* (bool): True if data is valid. False otherwise.
- *reason* (Iterable): List of strings, indicating validation errors. Empty list in case if *result* is True.

12.5.1.11 ska.utils module

class ska.utils.**RequestHelper**(*signature_param: str = 'signature'*, *auth_user_param: str = 'auth_user'*, *valid_until_param: str = 'valid_until'*, *extra_param: str = 'extra'*, *signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>*)

Bases: `object`

Request helper for easy put/extract of signature params from URLs.

extract_signed_data(*data: Dict[str, bytes | str | float | int]*, *secret_key: str | None = None*, *validate: bool = False*, *fail_silently: bool = False*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → Dict[str, str]

Extract signed data from the request.

Parameters

- **data** –
- **secret_key** –
- **validate** –
- **fail_silently** –
- **value_dumper** –
- **quoter** –

Returns

signature_to_dict(*signature:* AbstractSignature) → Dict[str, bytes | str | float | int]

Put signature into a dictionary.

Dictionary can be used later on to send requests (for example, a POST request) to the server.

Parameters
signature – Signature class.

Returns

Example

Required imports.

```
>>> from ska import Signature, RequestHelper
```

Generate signature.

```
>>> signature = Signature.generate_signature(
>>>     auth_user='user',
>>>     secret_key='your-secret-key'
>>> )
```

Create a request helper.

```
>>> request_helper = RequestHelper(
>>>     signature_param='signature',
>>>     auth_user_param='auth_user',
>>>     valid_until_param='valid_until'
>>> )
```

Appending signature params to the endpoint URL.

```
>>> signed_dict = request_helper.signature_to_dict(
>>>     signature=signature
>>> )
{
    'signature': 'YlZpLFsjUKBalL4x5trhkeEgqE8=',
    'auth_user': 'user',
    'valid_until': '1378045287.0'
}
```

signature_to_url(*signature: AbstractSignature*, *endpoint_url: str = ''*, *suffix: str = '?'*) → str

URL encodes the signature params.

Parameters

- **signature** – Signature class.
- **endpoint_url** –
- **suffix** – Suffix to add after the `endpoint_url` and before the appended signature params.

Returns

Example

Required imports.

```
>>> from ska import Signature, RequestHelper
```

Generate signature.

```
>>> signature = Signature.generate_signature(
>>>     auth_user='user',
>>>     secret_key='your-secret-key'
>>> )
```

Create a request helper.

```
>>> request_helper = RequestHelper(
>>>     signature_param='signature',
>>>     auth_user_param='auth_user',
>>>     valid_until_param='valid_until'
>>> )
```

Appending signature params to the endpoint URL.

```
>>> url = request_helper.signature_to_url(
>>>     signature=signature,
>>>     endpoint_url='http://e.com/api/'
>>> )
http://e.com/api/?valid_until=1378045287.0&auth_user=user&
↪signature=YlZpLFsjUKBalL4x5trhkeEgqE8%3D
```

`validate_request_data`(*data: Dict[str, bytes | str | float | int], secret_key: str, value_dumper: Callable | None = None, quoter: Callable | None = None*) → *SignatureValidationResult*

Validate the request data.

Parameters

- **`data`** –
- **`secret_key`** –
- **`value_dumper`** –
- **`quoter`** –

Returns

Example

If your imaginary *HttpRequest* object has *GET* property (dict), then you would validate the request data as follows.

Create a *RequestHelper* object with param names expected.

Required imports.

```
>>> from ska import RequestHelper
```

Create a request helper.

```
>>> request_helper = RequestHelper(
>>>     signature_param='signature',
>>>     auth_user_param='auth_user',
>>>     valid_until_param='valid_until'
>>> )
```

Validate the request data.

```
>>> validation_result = request_helper.validate_request_data(
>>>     data=request.GET,
>>>     secret_key='your-secret-key'
>>> )
```

12.5.1.12 Module contents

class ska.**HMACMD5Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC MD5 signature.

auth_user

extra

classmethod make_hash(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

class ska.**HMACSHA224Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC SHA-224 signature.

auth_user

extra

classmethod make_hash(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

class ska.**HMACSHA256Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC SHA-256 signature.

auth_user

extra

classmethod **make_hash**(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **auth_user** –
- **secret_key** –
- **valid_until** – Unix timestamp, valid until.
- **extra** – Additional variables to be added.
- **value_dumper** –
- **quoter** –

Returns

signature

valid_until

class ska.**HMACSHA384Signature**(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *AbstractSignature*

HMAC SHA-384 signature.

auth_user

`extra`

`classmethod make_hash`(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **`auth_user`** –
- **`secret_key`** –
- **`valid_until`** – Unix timestamp, valid until.
- **`extra`** – Additional variables to be added.
- **`value_dumper`** –
- **`quoter`** –

Returns

`signature`

`valid_until`

`class ska.HMACSHA512Signature`(*signature: bytes*, *auth_user: str*, *valid_until: float | str*, *extra: Dict[str, bytes | str | float | int] | None = None*)

Bases: *`AbstractSignature`*

HMAC SHA-512 signature.

`auth_user`

`extra`

`classmethod make_hash`(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *extra: Dict[str, bytes | str | float | int] | None = None*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → bytes

Make hash.

You should implement this method in your signature class.

Parameters

- **`auth_user`** –
- **`secret_key`** –
- **`valid_until`** – Unix timestamp, valid until.
- **`extra`** – Additional variables to be added.
- **`value_dumper`** –
- **`quoter`** –

Returns

`signature`

`valid_until`

class **ska.RequestHelper**(*signature_param: str = 'signature'*, *auth_user_param: str = 'auth_user'*, *valid_until_param: str = 'valid_until'*, *extra_param: str = 'extra'*, *signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>*)

Bases: `object`

Request helper for easy put/extract of signature params from URLs.

extract_signed_data(*data: Dict[str, bytes | str | float | int]*, *secret_key: str | None = None*, *validate: bool = False*, *fail_silently: bool = False*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → Dict[str, str]

Extract signed data from the request.

Parameters

- **data** –
- **secret_key** –
- **validate** –
- **fail_silently** –
- **value_dumper** –
- **quoter** –

Returns

signature_to_dict(*signature: AbstractSignature*) → Dict[str, bytes | str | float | int]

Put signature into a dictionary.

Dictionary can be used later on to send requests (for example, a POST request) to the server.

Parameters
signature – Signature class.

Returns

Example

Required imports.

```
>>> from ska import Signature, RequestHelper
```

Generate signature.

```
>>> signature = Signature.generate_signature(
>>>     auth_user='user',
>>>     secret_key='your-secret-key'
>>> )
```

Create a request helper.

```
>>> request_helper = RequestHelper(
>>>     signature_param='signature',
>>>     auth_user_param='auth_user',
>>>     valid_until_param='valid_until'
>>> )
```

Appending signature params to the endpoint URL.

```
>>> signed_dict = request_helper.signature_to_dict(
>>>     signature=signature
>>> )
{
    'signature': 'YlZpLFsjUKBalL4x5trhkeEgqE8=',
    'auth_user': 'user',
    'valid_until': '1378045287.0'
}
```

signature_to_url(*signature: AbstractSignature*, *endpoint_url: str = ''*, *suffix: str = '?'*) → str

URL encodes the signature params.

Parameters

- **signature** – Signature class.
- **endpoint_url** –
- **suffix** – Suffix to add after the `endpoint_url` and before the appended signature params.

Returns

Example

Required imports.

```
>>> from ska import Signature, RequestHelper
```

Generate signature.

```
>>> signature = Signature.generate_signature(
>>>     auth_user='user',
>>>     secret_key='your-secret-key'
>>> )
```

Create a request helper.

```
>>> request_helper = RequestHelper(
>>>     signature_param='signature',
>>>     auth_user_param='auth_user',
>>>     valid_until_param='valid_until'
>>> )
```

Appending signature params to the endpoint URL.

```
>>> url = request_helper.signature_to_url(
>>>     signature=signature,
>>>     endpoint_url='http://e.com/api/'
>>> )
http://e.com/api/?valid_until=1378045287.0&auth_user=user&
↪signature=YlZpLFsjUKBalL4x5trhkeEgqE8%3D
```

validate_request_data(*data: Dict[str, bytes | str | float | int]*, *secret_key: str*, *value_dumper: Callable | None = None*, *quoter: Callable | None = None*) → *SignatureValidationResult*

Validate the request data.

Parameters

- **data** –
- **secret_key** –
- **value_dumper** –
- **quoter** –

Returns

Example

If your imaginary *HttpRequest* object has *GET* property (dict), then you would validate the request data as follows.

Create a *RequestHelper* object with param names expected.

Required imports.

```
>>> from ska import RequestHelper
```

Create a request helper.

```
>>> request_helper = RequestHelper(
>>>     signature_param='signature',
>>>     auth_user_param='auth_user',
>>>     valid_until_param='valid_until'
>>> )
```

Validate the request data.

```
>>> validation_result = request_helper.validate_request_data(
>>>     data=request.GET,
>>>     secret_key='your-secret-key'
>>> )
```

ska.**Signature**

alias of *HMACSHA1Signature*

class ska.**SignatureValidationResult**(*result: bool*, *errors: List[ErrorCode | Any] | None = None*)

Bases: `object`

Signature validation result container.

If signature validation result is True, things like this would work:

```
>>> res = SignatureValidationResult(result=True)
>>> print bool(res)
True
>>> res = SignatureValidationResult(
>>>     result=False,
>>>     reason=[error_codes.INVALID_SIGNATURE,]
>>> )
>>> print bool(res)
False
```

property message: str

Human readable message of all errors.

Returns

`property reason: map`

Reason.

For backwards compatibility. Returns list of text messages.

Returns

`ska.extract_signed_request_data`(*data: ~typing.Dict[str, bytes | str | float | int], secret_key: str | None = None, signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra_param: str = 'extra', validate: bool = False, fail_silently: bool = False, signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>, value_dumper: ~typing.Callable | None = None, quoter: ~typing.Callable | None = None*) → Dict[str, bytes | str | float | int]

Validate the signed request data.

Parameters

- **data** – Dictionary holding the (HTTP) request (for example GET or POST) data.
- **secret_key** – The shared secret key.
- **signature_param** – Name of the (for example GET or POST) param name which holds the `signature` value.
- **auth_user_param** – Name of the (for example GET or POST) param name which holds the `auth_user` value.
- **valid_until_param** – Name of the (foe example GET or POST) param name which holds the `valid_until` value.
- **extra_param** – Name of the (foe example GET or POST) param name which holds the `extra` value.
- **validate** – If set to True, request data is validated before returning the result.
- **fail_silently** – If set to True, exceptions are omitted.
- **signature_cls** –
- **value_dumper** –
- **quoter** –

Returns

Dictionary with signed request data.

`ska.sign_url`(*auth_user: str, secret_key: str, valid_until: float | str | None = None, lifetime: int = 600, url: str = '', suffix: str = '?', signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra: ~typing.Dict[str, bytes | str | float | int] | None = None, extra_param: str = 'extra', signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>, value_dumper: ~typing.Callable | None = None*) → str

Sign the URL.

Parameters

- **auth_user** – Username of the user making the request.
- **secret_key** – The shared secret key.
- **valid_until** – Unix timestamp. If not given, generated automatically (now + lifetime).

- **lifetime** – Signature lifetime in seconds.
- **url** – URL to be signed.
- **suffix** – Suffix to add after the `endpoint_url` and before the appended signature params.
- **signature_param** – Name of the GET param name which would hold the generated signature value.
- **auth_user_param** – Name of the GET param name which would hold the `auth_user` value.
- **valid_until_param** – Name of the GET param name which would hold the `valid_until` value.
- **extra** – Extra variables to add to the request.
- **extra_param** – Name of the GET param name which would hold the `extra_keys` value.
- **signature_cls** –
- **value_dumper** –

Returns

Example

Required imports.

```
>>> from ska import sign_url
```

Producing a signed URL.

```
>>> signed_url = sign_url(
>>>     auth_user='user', secret_key='your-secret_key', lifetime=120,
>>>     url='http://e.com/api/', signature_param=DEFAULT_SIGNATURE_PARAM,
>>>     auth_user_param=DEFAULT_AUTH_USER_PARAM,
>>>     valid_until_param=DEFAULT_VALID_UNTIL_PARAM,
>>>     extra={
>>>         'provider': 'service1.example.com',
>>>         'email': 'john.doe@mail.example.com'
>>>     },
>>>     extra_param = DEFAULT_EXTRA_PARAM
>>> )
http://e.com/api/?valid_until=1378045287.0&auth_user=user&signature=
YlZpLFsjUKBalL4x5trhkeEgqE8%3D
```

ska.**signature_to_dict**(*auth_user: str*, *secret_key: str*, *valid_until: float | str | None = None*, *lifetime: int = 600*, *signature_param: str = 'signature'*, *auth_user_param: str = 'auth_user'*, *valid_until_param: str = 'valid_until'*, *extra: ~typing.Dict[str, str | int] | None = None*, *extra_param: str = 'extra'*, *signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>*, *value_dumper: ~typing.Callable | None = None*, *quoter: ~typing.Callable | None = None*) → Dict[str, bytes | str | float | int]

Return a dictionary containing the signature data params.

Parameters

- **auth_user** – Username of the user making the request.
- **secret_key** – The shared secret key.

- **valid_until** – Unix timestamp. If not given, generated automatically (now + lifetime).
- **lifetime** – Signature lifetime in seconds.
- **signature_param** – Name of the (for example POST) param name which would hold the generated `signature` value.
- **auth_user_param** – Name of the (for example POST) param name which would hold the `auth_user` value.
- **valid_until_param** – Name of the (for example POST) param name which would hold the `valid_until` value.
- **extra** – Additional arguments for the signature.
- **extra_param** – Name of the (for example POST) param name which would hold the `extra` keys value.
- **signature_cls** –
- **value_dumper** –
- **quoter** –

Returns

Example

Required imports.

```
>>> from ska import signature_to_dict
```

Producing a dictionary with signature data.

```
>>> signature_dict = signature_to_dict(
>>>     auth_user='user', secret_key='your-secret_key', lifetime=120,
>>>     signature_param=DEFAULT_SIGNATURE_PARAM,
>>>     auth_user_param=DEFAULT_AUTH_USER_PARAM,
>>>     valid_until_param=DEFAULT_VALID_UNTIL_PARAM
>>> )
{
    'signature': 'YlZpLFsjUKBalL4x5trhkeEgqE8=',
    'auth_user': 'user',
    'valid_until': '1378045287.0'
}
```

`ska.`**`validate_signed_request_data`**(*data: ~typing.Dict[str, bytes | str | float | int], secret_key: str, signature_param: str = 'signature', auth_user_param: str = 'auth_user', valid_until_param: str = 'valid_until', extra_param: str = 'extra', signature_cls: ~typing.Type[~ska.base.AbstractSignature] = <class 'ska.signatures.hmac_sha1.HMACSHA1Signature'>, value_dumper: ~typing.Callable | None = None, quoter: ~typing.Callable | None = None*) → *SignatureValidationResult*

Validate the signed request data.

Parameters

- **data** – Dictionary holding the (HTTP) request (for example GET or POST) data.
- **secret_key** – The shared secret key.

- **`signature_param`** – Name of the (for example GET or POST) param name which holds the `signature` value.
- **`auth_user_param`** – Name of the (for example GET or POST) param name which holds the `auth_user` value.
- **`valid_until_param`** – Name of the (foe example GET or POST) param name which holds the `valid_until` value.
- **`extra_param`** – Name of the (foe example GET or POST) param name which holds the `extra` keys value.
- **`signature_cls`** –
- **`value_dumper`** –
- **`quoter`** –

Returns

A `ska.SignatureValidationResult` object with the following properties:

- *result* (bool): True if data is valid. False otherwise.
- *reason* (Iterable): List of strings, indicating validation errors. Empty list in case if *result* is True.

12.6 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX