Signing & Unsigning

**20th july**

IDOR attack (Insecure direct object reference) from url hardcoded credentials can be accessed by any other user if it visible in url.

to protect from it we use Django Signer.

Signing and unsigning . Signing displays the context in alphabatic format

and to get it in orginal format we use unsigning.

**Signing & unsigning is used on string.**

1. First we need to import the django.core.signing import Signer

2. Then we need to instantiate the object

signer = Signer()

3. Then pass the value to be sign in function

signer = signer.sign("string value")

**For list, tuple, dictionary sign\_object is used.**

1. First we need to import the django.core.signing import Signer

2. Then we need to instantiate the object

signer = Signer()

3. Then pass the value to be sign in function

signed\_obj = signer.sign\_object({"message": "Hello!"})

# Cryptographic signing

You may also find signing useful for the following:

* Generating “recover my account” URLs for sending to users who have lost their password.
* Ensuring data stored in hidden form fields has not been tampered with.
* Generating one-time secret URLs for allowing temporary access to a protected resource, for example a downloadable file that a user has paid for.

## **Using the low-level API**[**¶**](https://docs.djangoproject.com/en/4.2/topics/signing/#using-the-low-level-api)

1. Django’s signing methods live in the **django.core.signing** module. To sign a value, first instantiate a **Signer** instance:

**>>> from** **django.core.signing** **import** Signer

**>>>** signer = Signer()

**>>>** value = signer.sign("My string")

**>>>** value

'My string:GdMGD6HNQ\_qdgxYP8yBZAdAIV1w'

1. The signature is appended to the end of the string, following the colon. You can retrieve the original value using the **unsign** method:

**>>>** original = signer.unsign(value)

**>>>** original

'My string'

1. If you pass a non-string value to **sign**, the value will be forced to string before being signed, and the **unsign** result will give you that string value:

**>>>** signed = signer.sign(2.5)

**>>>** original = signer.unsign(signed)

**>>>** original

'2.5'

1. If you wish to protect a list, tuple, or dictionary you can do so using the **sign\_object()** and **unsign\_object()** methods:

**>>>** signed\_obj= signer.sign\_object({"message": "Hello!"})

**>>>** signed\_obj

'eyJtZXNzYWdlIjoiSGVsbG8hIn0:Xdc-mOFDjs22KsQAqfVfi8PQSPdo3ckWJxPWwQOFhR4'

**>>>** obj = signer.unsign\_object(signed\_obj)

**>>>** obj

{'message': 'Hello!'}

For signing (encrypting) use dumps

For unsigning (unencrypting) use loads

**>>> from** **django.core** **import** signing

**>>>** signer = signing.TimestampSigner()

**>>>** value = signer.sign\_object({"foo": "bar"})

**>>>** value

'eyJmb28iOiJiYXIifQ:1kx6R3:D4qGKiptAqo5QW9iv4eNLc6xl4RwiFfes6oOcYhkYnc'

**>>>** signer.unsign\_object(value)

{'foo': 'bar'}

**>>>** value = signing.dumps({"foo": "bar"})

**>>>** value

'eyJmb28iOiJiYXIifQ:1kx6Rf:LBB39RQmME-SRvilheUe5EmPYRbuDBgQp2tCAi7KGLk'

**>>>** signing.loads(value)

{'foo': 'bar'}

References:

https://docs.djangoproject.com/en/4.2/topics/signing/