pyUmbral Documentation

Release 0.1.3-alpha.1

Michael Egorov, Justin Myles Holmes, David Nuñez, John Pacific,

May 21, 2019

Table of Contents:

pyUmbral is the reference implementation of the Umbral threshold proxy re-encryption scheme. It is open-source, built with Python, and uses OpenSSL and Cryptography.io.

Using Umbral, Alice (the data owner) can *delegate decryption rights* to Bob for any ciphertext intended to her, through a re-encryption process performed by a set of semi-trusted proxies or *Ursulas*. When a threshold of these proxies participate by performing re-encryption, Bob is able to combine these independent re-encryptions and decrypt the original message using his private key.

pyUmbral is the cryptographic engine behind nucypher, a proxy re-encryption network to empower privacy in decentralized systems.

CHAPTER 1

Installing pyUmbral

v0.1.3-alpha.1

1.1 Using pip

The easiest way to install pyUmbral is using `pip`:

```
$ pip3 install umbral
```

1.2 Build from source code

pyUmbral is maintained on GitHub: https://github.com/nucypher/pyUmbral.

Clone the repository to download the source code.

```
$ git clone https://github.com/nucypher/pyUmbral.git
```

Once you have acquired the source code, you can...

... embed pyUmbral modules into your own codebase...

```
from umbral import pre, keys, config
```

... install pyUmbral with pipenv...

```
$ pipenv install .
```

... or install it with python-pip...

```
$ pip3 install .
```

1.3 Install dependencies

The NuCypher team uses pipenv for managing pyUmbral's dependencies. The recommended installation procedure is as follows:

```
$ sudo pip3 install pipenv
$ pipenv install
```

Post-installation, you can activate the pyUmbral's virtual environment in your current terminal session by running `pipenv shell`.

If your installation is successful, the following command will succeed without error.

```
$ pipenv run python
>>> import umbral
```

For more information on pipenv, The official documentation is located here: https://docs.pipenv.org/.

1.4 Development Installation

If you want to participate in developing pyUmbral, you'll probably want to run the test suite and / or build the documentation, and for that, you must install some additional development requirements.

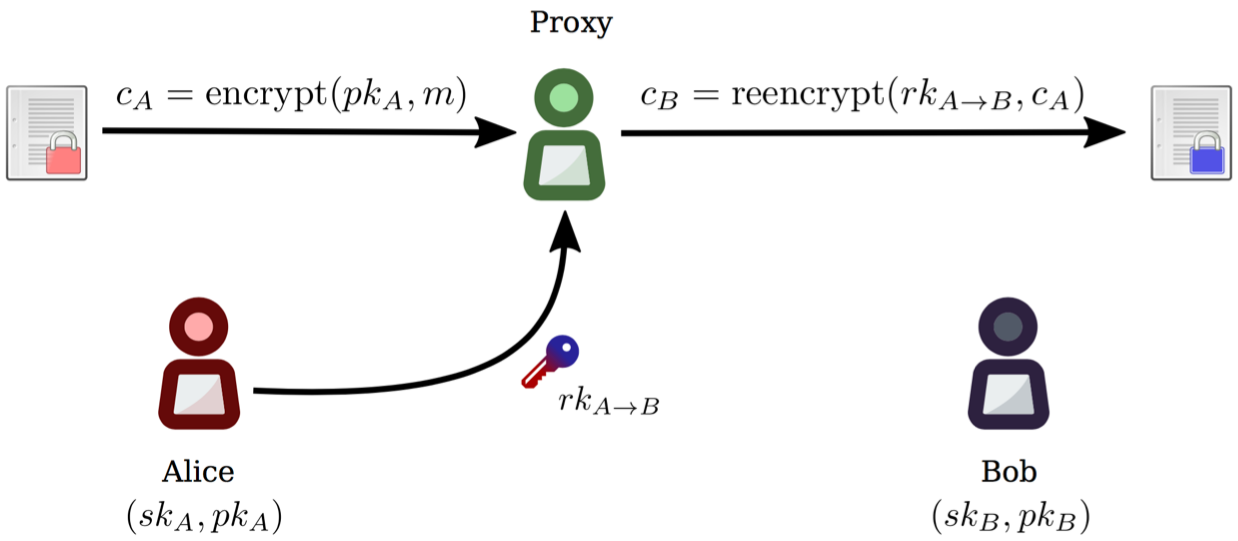
```
$ pipenv install --dev --three
```

To build the documentation locally:

```
$ pipenv run make html --directory=docs
```

CHAPTER 2

Using pyUmbral



2.1 Configuration

2.1.1 Setting the default curve

The best way to start using pyUmbral is to decide on an elliptic curve to use and set it as your default.

```
>>> from umbral import config
>>> from umbral.curve import SECP256K1
>>> config.set_default_curve(SECP256K1)
```

For more information on curves, see choosing_and_using_curves.

2.2 Encryption

2.2.1 Generate an Umbral key pair

First, let's generate two asymmetric key pairs for Alice: A delegating key pair and a signing key pair.

```
>>> from umbral import keys, signing

>>> alices_private_key = keys.UmbralPrivateKey.gen_key()
>>> alices_public_key = alices_private_key.get_pubkey()

>>> alices_signing_key = keys.UmbralPrivateKey.gen_key()
>>> alices_verifying_key = alices_signing_key.get_pubkey()
>>> alices_signer = signing.Signer(private_key=alices_signing_key)
```

2.2.2 Encrypt with a public key

Now let's encrypt data with Alice's public key. Invocation of `pre.encrypt` returns both the `ciphertext` and a `capsule`. Note that anyone with Alice's public key can perform this operation.

```
>>> from umbral import pre
>>> plaintext = b'Proxy Re-encryption is cool!'
>>> ciphertext, capsule = pre.encrypt(alices_public_key, plaintext)
```

2.2.3 Decrypt with a private key

Since data was encrypted with Alice's public key, Alice can open the capsule and decrypt the ciphertext with her private key.

```
>>> cleartext = pre.decrypt(ciphertext=ciphertext,
...                         capsule=capsule,
...                         decrypting_key=alices_private_key)
```

2.3 Threshold Re-Encryption

2.3.1 Bob Exists

```
>>> from umbral import keys
>>> bobs_private_key = keys.UmbralPrivateKey.gen_key()
>>> bobs_public_key = bobs_private_key.get_pubkey()
```

2.3.2 Alice grants access to Bob by generating kfrags

When Alice wants to grant Bob access to open her encrypted messages, she creates *re-encryption key fragments*, or *"kfrags"*, which are next sent to N proxies or *Ursulas*.

Alice must specify `N` (the total number of kfrags), and a `threshold` (the minimum number of kfrags needed to activate a capsule). In the following example, Alice creates 20 kfrags, but Bob needs to get only 10 re-encryptions to activate the capsule.

```
>>> kfrags = pre.generate_kfrags(delegating_privkey=alices_private_key,
...                              signer=alices_signer,
...                              receiving_pubkey=bobs_public_key,
...                              threshold=10,
...                              N=20)
```

2.3.3 Bob receives a capsule

Next, let's generate a key pair for Bob, and pretend to send him the capsule through a side channel like S3, IPFS, Google Cloud, Sneakernet, etc.

```
# Bob receives the capsule through a side-channel: IPFS, Sneakernet, etc.
capsule = <fetch the capsule through a side-channel>
```

2.3.4 Bob fails to open the capsule

If Bob attempts to open a capsule that was not encrypted for his public key, or re-encrypted for him by Ursula, he will not be able to open it.

```
>>> fail = pre.decrypt(ciphertext=ciphertext,
...                    capsule=capsule,
...                    decrypting_key=bobs_private_key)
Traceback (most recent call last):
    ...
umbral.pre.UmbralDecryptionError
```

2.3.5 Ursulas perform re-encryption

Bob asks several Ursulas to re-encrypt the capsule so he can open it. Each Ursula performs re-encryption on the capsule using the `kfrag` provided by Alice, obtaining this way a "capsule fragment", or `cfrag`. Let's mock a network or transport layer by sampling `threshold` random kfrags, one for each required Ursula. Note that each Ursula must prepare the received capsule before re-encryption by setting the proper correctness keys.

Bob collects the resulting cfrags from several Ursulas. Bob must gather at least `threshold` cfrags in order to activate the capsule.

```
>>> import random
>>> kfrags = random.sample(kfrags,  # All kfrags from above
...                        10)      # M - Threshold

>>> capsule.set_correctness_keys(delegating=alices_public_key,
...                              receiving=bobs_public_key,
...                              verifying=alices_verifying_key)
(True, True, True)

>>> cfrags = list()                 # Bob's cfrag collection
>>> for kfrag in kfrags:
...     cfrag = pre.reencrypt(kfrag=kfrag, capsule=capsule)
...     cfrags.append(cfrag)        # Bob collects a cfrag
```

2.4 Decryption

2.4.1 Bob attaches cfrags to the capsule

Bob attaches at least `threshold` cfrags to the capsule, which has to be prepared in advance with the necessary correctness keys. Only then it can become *activated*.

```
>>> capsule.set_correctness_keys(delegating=alices_public_key,
...                              receiving=bobs_public_key,
...                              verifying=alices_verifying_key)
(False, False, False)

>>> for cfrag in cfrags:
...     capsule.attach_cfrag(cfrag)
```

2.4.2 Bob activates and opens the capsule

Finally, Bob decrypts the re-encrypted ciphertext using the activated capsule.

```
>>> cleartext = pre.decrypt(ciphertext=ciphertext,
...                         capsule=capsule,
...                         decrypting_key=bobs_private_key)
```

CHAPTER 3

Academic Whitepaper

The Umbral scheme academic whitepaper and cryptographic specifications are available on GitHub.

> "Umbral: A Threshold Proxy Re-Encryption Scheme" *by David Nuñez*. https://github.com/nucypher/umbral-doc/blob/master/umbral-doc.pdf

CHAPTER 4

Support & Contribute

- Issue Tracker: https://github.com/nucypher/pyUmbral/issues
- Source Code: https://github.com/nucypher/pyUmbral

CHAPTER 5

Security

If you identify vulnerabilities with _any_ nucypher code, please email security@nucypher.com with relevant information to your findings. We will work with researchers to coordinate vulnerability disclosure between our partners and users to ensure successful mitigation of vulnerabilities.

Throughout the reporting process, we expect researchers to honor an embargo period that may vary depending on the severity of the disclosure. This ensures that we have the opportunity to fix any issues, identify further issues (if any), and inform our users.

Sometimes vulnerabilities are of a more sensitive nature and require extra precautions. We are happy to work together to use a more secure medium, such as Signal. Email security@nucypher.com and we will coordinate a communication channel that we're both comfortable with.

CHAPTER 6

Indices and Tables

- genindex
- modindex
- search