nthcommunity

Release 1.0.0

Nth Party, Ltd.

Apr 17, 2022

CONTENTS

1 Purpose and Features **3**

2 Package Installation and Usage **5**
2.1 Example: Secure Intersection Size 5

3 Documentation **9**
3.1 nthcommunity module 9

4 Testing and Conventions **15**

5 Contributions **17**

6 Versioning **19**

7 Publishing **21**

Python Module Index **23**

Index **25**

Open-source Python library that allows developers to leverage the nth.community service platform and API to implement secure, privacy-preserving data collaborations within their web services and applications.

CHAPTER

ONE

PURPOSE AND FEATURES

This library is a client-side component and Python API for the nth.community secure, privacy-preserving data collaboration service platform. Together, this library and the nth.community service platform make it possible to define and execute data workflows (called *collaborations*) that operate on encrypted input data without decrypting it.

This open-source library supports a very limited set of input data types (non-negative 32-bit integers and single-column tables of strings) and operations (intersection of tables, row count of an intersection of tables, and summation of integers). The secure data collaboration workflows enabled by nth.community protect contributor inputs by relying on secure multi-party computation protocols, including private set intersection (via the oblivious library) and additive secret sharing (via the additive library).

Designed to be integrated easily into full-stack web applications, this library organizes secure multi-party computation workflows into a familiar structure that resembles public-key cryptographic systems. In order to improve portability of data and to allow programmers to leverage native Python features, data structures in the library are derived from built-in Python types and can be converted in a straightforward way to and from ubiquitous formats such as JSON.

CHAPTER

TWO

PACKAGE INSTALLATION AND USAGE

The package is available on PyPI:

```
python -m pip install nthcommunity
```

The library can be imported in the usual ways:

```
import nthcommunity
from nthcommunity import *
```

2.1 Example: Secure Intersection Size

A secure, privacy-preserving data collaboration begins when a recipient party is created. This party is responsible for defining the collaboration data workflow, distributing contribution keys to contributors, and performing the computation on the encrypted data contributions in concert with the nth.community service platform:

```
>>> from nthcommunity import *
>>> r = recipient()
```

Individual contributor objects encapsulate individual contributors. Two contributors are defined below:

```
>>> c_a = contributor()
>>> c_b = contributor()
```

Each contributor is assigned a unique identifier when the `contributor` object is created:

```
>>> c_a.identifier()
'57728767-55bf-4833-ab36-d3733a0e8448'
```

A collaboration is a tree-like data structure that defines how the overall result of a collaborative data workflow is computed. Every leaf node in the data structure must indicate which contributor must supply the encrypted data corresponding to that node. The internal, non-leaf nodes represent data operations such as `count`, `intersection`, and `summation`. The collaboration workflow defined below computes the size of the intersection between two tables (one from each contributor):

```
>>> w = count(intersection(table(contributor=c_a), table(contributor=c_b)))
```

The recipient can (by leveraging the nth.community service platform API) generate a dictionary that maps each of the contributor identifiers to their respective contribution key:

```
>>> id_to_key = r.generate(w)
```

Each of these individual keys `id_to_key[c_a.identifier()]` and `id_to_key[c_b.identifier()]` can be delivered to their corresponding contributor. Note that the `collaboration` class is derived from `dict`, making conversion to JSON straightforward (using either the built-in json library or the wrapper method used below):

```
>>> print(id_to_key[c_a.identifier()].to_json(indent=2))
{
  "type": "operation",
  "value": "count",
  "arguments": [
    {
      "type": "operation",
      "value": "intersection",
      "arguments": [
        {
          "type": "table",
          "contributor": {
            "type": "contributor",
            "identifier": "6a8dd844-6003-4475-aea7-b4182400bb90"
          },
          "public": "tdeJzrPIEpsFcykqnlN4M/hiP8gnNAJiSBRysLIutwo="
        }
      ]
    }
  ],
  "material": {
    "public": "F/RIe6jQ38Uk1CUdW7JKwd7Q9b+J+HlnOARSM70zERg=",
    "scalar": "A4f8MRxKsxZnKiScaJPw/O6uPzqBfREeNaPAaOdlnAU="
  },
  "certificate": "U4HFZcQ1hCplmec50gEDYnMxHn/
↪ILclPmR6KC04uz1i5sJBwpEq0HB+tMlRPzWZu6dpPNuAjOzlIOshSgw7EXA=="
}
```

Each contributor can then encrypt their data contribution using their key:

```
>>> table_a = [['a'], ['b'], ['c'], ['d']]
>>> enc_a = c_a.encrypt(id_to_key[c_a.identifier()], table_a)
>>> table_b = [['b'], ['c'], ['d'], ['e']]
>>> enc_b = c_b.encrypt(id_to_key[c_b.identifier()], table_b)
```

As with contributor keys, encrypted contributions can be converted to and from JSON in a straightforward way:

```
>>> payload = enc_a.to_json(indent=2)
>>> print(payload)
{
  "type": "operation",
  "value": "count",
  "arguments": [
    {
      "type": "operation",
      "value": "intersection",
      "arguments": [
```

(continues on next page)

(continued from previous page)

```
        {
          "type": "table",
          "contributor": {
            "type": "contributor",
            "identifier": "6a8dd844-6003-4475-aea7-b4182400bb90"
          },
          "public": "tdeJzrPIEpsFcykqnlN4M/hiP8gnNAJiSBRysLIutwo=",
          "value": [
            [
              "0Ch4F9+RVGu+e0dmK+ydFvOZSLld/xMk4xcXX8U2TGY5/
↪CUiwCtPb0bM6nwJrbZPVBIAWa5tgVCGR2Tu8LdfalxDwpGqyNuWPcUtQP8T6P8=",

↪"F1CZ5kRyYpEoZhlwTLu1gShLsH1HDZxQnQnypgDncyG+HGCrHn+UtKb9B4bvDaxbb8Nl92TV+U5ouE8MePbnFkslfwZyD6g5hWh4
↪"
            ],
            [

↪"4noLjc9aSpY1owBlSHv9IYZr6bvxkQUhhKOhV9Ty2kZTJP7UbXgSyxjsqEr1qbEWXQ1vP1uWS+JMN+rWqtpblcitqcg/
↪epuPYWvJi6vr9vE=",
              "JC1L0t4JeVDr/+40c/
↪JAsnpTqQUZLELZ5hJukDlX1Uxxf5MqZXglNc1aR9p+A5fvIZPj1P3YHOMS1Cbjpy1R3iZNUMsTOkn7pjhkZdkGE+XzHVRKz9TQdmk
↪"
            ],
            [

↪"TmQcw0iFgnnUofuJj4hhof6oqNp1iQ94E+jqNCdQT3i8kcL4D7I33lJdLyiUwAQHm1BE6SrU+ZS+jOrfle5mfY91PuuxFsB6UBlc
↪",
              "puSkEfdRdNiIO668GG5jwV/zp84OhFN9rs5UjOoYNEUHif1BzrNDMq8huDJxFW8k78GbjMPZ/
↪D4Yn5S3bjg/CwmGFPb99o4p7OwBYP1pGuds8smzUaVqcPt5FCCJxZWGYQ=="
            ],
            [

↪"PcBEJUe6IjiJ5EKUBTfeebzlLo2s291sAEdi9UnSQ3O1yg+ttOVxQQNN2QVvc4V8Swj6J+FBbC+S3+av1yV/
↪57yW88J8YJYBTLlK0awWYqw=",
              "ubwDp7Up8bRvznbOSmKlTDJ4MlC1JIz8zFaOHQIdfiCK1LPCKeyqIOaX/
↪BPRbmNUkGQA9raLdsgUvWH9TQzGd3c7nvmbjltL9e4WWofM72+fBdj+/Dgg16+YAdEY6fXoog=="
            ]
          ]
        }
      ]
    }
  ]
}
>>> collaboration.from_json(payload) is not None
True
```

Once all the encrypted contributions are received, the recipient can (by again leveraging the nth.community service platform API) execute the workflow on the encrypted contributions to obtain a result:

```
>>> result = r.evaluate({c_a.identifier(): enc_a, c_b.identifier(): enc_b})
>>> result["value"]
3
```

CHAPTER

THREE

DOCUMENTATION

The library contains one module:

3.1 nthcommunity module

Python library for the nth.community secure, privacy-preserving data collaboration service platform and API.

An end-to-end example is presented in the documentation for the *recipient* class.

class nthcommunity.nthcommunity.**collaboration**

Bases: dict

Base class for tree data structure representing a data collaboration. Consult definitions of derived classes such as *intersection* and *table* for more details, and find in-context usage examples in the documentation for the *recipient* class.

static from_json(*argument: Union[str, dict]*) → *nthcommunity.nthcommunity.collaboration*

Parse a dictionary or JSON string into an instance of this class. Find usage examples in the documentation for the *recipient* class.

to_json(**args*, ***kwargs*) → str

Convert an instance of this class into a JSON string. This method is a wrapper for the `json.dumps` method found in the built-in json library. Find usage examples in the documentation for the *recipient* class.

class nthcommunity.nthcommunity.**recipient**

Bases: object

A collaboration begins with a recipient party represented by a *recipient* object. The recipient defines a *collaboration* workflow, generates keys for individual contributors, accepts encrypted contributions, and computes the results in concert with the nth.community service platform.

```
>>> r = recipient()
```

Individual *contributor* objects represent and uniquely identify individual contributors.

```
>>> c_a = contributor()
>>> c_b = contributor()
```

The *contributor* class is derived from `dict` and contains a value under the `'identifier'` key that corresponds to the unique identifier for that contributor. This value can also be retrieved using the *contributor.identifier* method.

```
>>> 'identifier' in c_a
True
>>> c_a.identifier() is not None
True
```

A *collaboration* is a tree-like data structure that defines how the overall result of the collaboration is computed. Each leaf node of the data structure must indicate which contributor must supply the encrypted data corresponding to that node. The internal, non-leaf nodes represent data operations such as *count*, *summation*, and *intersection*.

```
>>> w = count(intersection(table(contributor=c_a), table(contributor=c_b)))
```

The recipient can use the *recipient.generate* method (which internally leverages the nth.community service platform) to create a dictionary that maps each contributor identifier to its respective contribution key *for that specific collaboration workflow*.

```
>>> id_to_key = r.generate(w)
```

Each of these individual keys `id_to_key[c_a.identifier()]` and `id_to_key[c_b.identifier()]` can be converted to JSON using the *collaboration.to_json* method and delivered to their corresponding contributor.

```
>>> key_a_json = id_to_key[c_a.identifier()].to_json()
>>> key_b_json = id_to_key[c_b.identifier()].to_json()
```

Contributors can parse the JSON strings using *collaboration.from_json*.

```
>>> key_a = collaboration.from_json(key_a_json)
>>> key_b = collaboration.from_json(key_b_json)
```

Each contributor can then use the *contributor.encrypt* method to encrypt their data contribution using their key. After validating the contribution key, this method *does not perform any external communications* and encrypts the input data *entirely within the host environment belonging to the contributor*. In particular, it does **not** communicate with the nth.community service platform or the recipient during this process.

```
>>> table_a = [['a'], ['b'], ['c'], ['d']]
>>> enc_a = c_a.encrypt(key_a, table_a)
>>> table_b = [['b'], ['c'], ['d'], ['e']]
>>> enc_b = c_b.encrypt(key_b, table_b)
```

Because encrypted contributions are also *collaboration* objects, they can be easily converted to and from JSON strings. The recipient can then use the *recipient.evaluate* method (which again internally leverages the nth.community service platform) to evaluate the encrypted contributions and obtain a result.

```
>>> result = r.evaluate({c_a.identifier(): enc_a, c_b.identifier(): enc_b})
>>> result["value"]
3
```

generate(*collaboration*) → dict

Submit a collaboration via the nth.community service platform API to receive the set of contributor keys that can be distributed to contributors. Find an in-context usage example in the documentation for the *recipient* class.

evaluate(*collaborations*) → *nthcommunity.nthcommunity.collaboration*

Evaluate a collaboration workflow instantiated with encrypted data contributions (represented as a dictio-

nary that maps each contributor's identifier to that contributor's encrypted contribution) by submitting it to the nth.community service platform. Find an in-context usage example in the documentation for the *recipient* class.

class nthcommunity.nthcommunity.**contributor**(*identifier=None*)

Bases: dict

Data structure and methods for an individual data contributor within a collaboration. A contributor receives a collaboration key from a recipient and encrypts their data contribution using that key.

```
>>> (r, c_a, c_b) = (recipient(), contributor(), contributor())
>>> w = count(intersection(table(contributor=c_a), table(contributor=c_b)))
>>> id_to_key = r.generate(w)
```

Each of the individual keys in `id_to_key.values()` can be delivered to their corresponding contributor. Each contributor can then use the key to encrypt their data contribution via the *contributor.encrypt* method, as shown below.

```
>>> id_a = c_a.identifier()
>>> table_a = [['a'], ['b'], ['c'], ['d']]
>>> table_a_encrypted = c_a.encrypt(id_to_key[id_a], table_a)
```

identifier() → str

Return this contributor's unique identifier.

```
>>> c = contributor()
>>> isinstance(c.identifier(), str)
True
```

static from_json(*argument: Union[str, dict]*) → *nthcommunity.nthcommunity.contributor*

Parse a dictionary or JSON string into an instance of this class.

```
>>> s = '{"type": "contributor", "identifier": "b1a63caf"}'
>>> isinstance(contributor.from_json(s), contributor)
True
```

to_json(**args*, ***kwargs*) → str

Convert an instance of this class into a JSON string. This method is a wrapper for the `json.dumps` method found in the built-in json library.

```
>>> c = contributor('b1a63caf-a549-429e-8ed0-44cd5fbb0eeb')
>>> c.to_json()
'{"type": "contributor", "identifier": "b1a63caf-a549-429e-8ed0-44cd5fbb0eeb"}'
```

validate(*collaboration*) → bool

Validate the certificate within a contribution key obtained from a recipient by submitting it to the nth.community service platform. The *contributor.encrypt* method automatically invokes this method before encrypting a contribution.

```
>>> (r, c_a, c_b) = (recipient(), contributor(), contributor())
>>> w = count(intersection(table(contributor=c_a), table(contributor=c_b)))
>>> key_a = r.generate(w)[c_a.identifier()]
>>> c_a.validate(key_a)
True
```

encrypt(*collaboration*, *contribution*) → *nthcommunity.nthcommunity.collaboration*

Encrypt a data set as a contribution to a collaboration. Find an in-context usage example in the documentation for the *recipient* class.

```
>>> (r, c_a, c_b) = (recipient(), contributor(), contributor())
>>> w = count(intersection(table(contributor=c_a), table(contributor=c_b)))
>>> key_a = r.generate(w)[c_a.identifier()]
>>> table_a = [['a'], ['b'], ['c'], ['d']]
>>> enc_a = c_a.encrypt(key_a, table_a)
```

After validating the contribution key, this method *does not perform any external communications* and encrypts the input data *entirely within the host environment belonging to the contributor*. In particular, it does **not** communicate with the nth.community service platform or the recipient during this process.

class nthcommunity.nthcommunity.**count**(**arguments*)

Bases: *nthcommunity.nthcommunity.collaboration*

Collaboration tree node for the count operation. A count operation can be applied to a single intersection collaboration.

```
>>> (c_a, c_b) = (contributor(), contributor())
>>> w = count(intersection(table(contributor=c_a), table(contributor=c_b)))
```

Any attempt to apply a count operation to a collaboration workflow that is not compatible with a count operation raises an exception.

```
>>> count(integer(123))
Traceback (most recent call last):
  ...
TypeError: count operation must be applied to a single intersection collaboration
```

class nthcommunity.nthcommunity.**summation**(**arguments*)

Bases: *nthcommunity.nthcommunity.collaboration*

Collaboration tree node for the summation operation. A summation operation can be applied to two or more contributors' integer contributions.

```
>>> (c_a, c_b) = (contributor(), contributor())
>>> w = summation(integer(contributor=c_a), integer(contributor=c_b))
```

Any attempt to apply a summation operation to a collaboration workflow that is not compatible with a summation operation raises an exception.

```
>>> summation(integer(123))
Traceback (most recent call last):
  ...
TypeError: summation operation must be applied to two or more integers
```

class nthcommunity.nthcommunity.**intersection**(**arguments*)

Bases: *nthcommunity.nthcommunity.collaboration*

Collaboration tree node for the intersection operation. An intersection operation can be applied to two or more contributors' table contributions.

```
>>> (c_a, c_b) = (contributor(), contributor())
>>> w = intersection(table(contributor=c_a), table(contributor=c_b))
```

Any attempt to apply an intersection operation to a collaboration workflow that is not compatible with an intersection operation raises an exception.

```
>>> intersection(table())
Traceback (most recent call last):
  ...
TypeError: intersection operation must be applied to two or more tables
```

class nthcommunity.nthcommunity.**integer**(*value=None*, *contributor=None*)

Bases: *nthcommunity.nthcommunity.collaboration*

Collaboration tree leaf node for a contributed integer value within a collaboration tree data structure. Only 32-bit non-negative integers are supported.

```
>>> c = contributor()
>>> w = integer(value=123, contributor=c)
```

Any attempt to construct an invalid integer contribution raises an exception.

```
>>> integer(-123)
Traceback (most recent call last):
  ...
ValueError: integer value must be a non-negative 32-bit integer
```

class nthcommunity.nthcommunity.**table**(*value=None*, *contributor=None*, *limit=None*)

Bases: *nthcommunity.nthcommunity.collaboration*

Collaboration tree leaf node for a contributed data table within a collaboration tree data structure. A contributed table must be a list of rows, and each row must a single-element list that contains exactly one string.

```
>>> c = contributor()
>>> w = table(value=[['a'], ['b'], ['c']], contributor=c)
```

Any attempt to construct a data table contribution that is invalid or that exceeds size limits raises an exception.

```
>>> table(['a', 'b', 'c'])
Traceback (most recent call last):
  ...
ValueError: table value must be a list of single-item lists, each containing a␣
↪string
>>> table([['a']] * 1001)
Traceback (most recent call last):
  ...
ValueError: table length exceeds maximum of 1000
>>> table([['a'  * 257]])
Traceback (most recent call last):
  ...
ValueError: table field value length exceeds maximum of 256
```

exception nthcommunity.nthcommunity.**ServiceError**

Bases: RuntimeError

Exception indicating that the service platform responded to an API request but indicated an error has occurred service-side (due to either an improperly configured service instance or an improper request).

```
>>> (c_a, c_b) = (contributor(), contributor())
>>> w = count(intersection(table(contributor=c_a), table(contributor=c_b)))
>>> del w["type"]
>>> try:
...     recipient().generate(w)
... except ServiceError as e:
...     print(e)
service did not return a valid response
```

The documentation can be generated automatically from the source files using Sphinx:

```
cd docs
python -m pip install -r requirements.txt
sphinx-apidoc -f -E --templatedir=_templates -o _source .. ../setup.py && make html
```

CHAPTER

FOUR

TESTING AND CONVENTIONS

All unit tests are executed and their coverage is measured when using pytest (see `setup.cfg` for configuration details):

```
python -m pip install pytest pytest-cov
python -m pytest
```

Style conventions are enforced using Pylint:

```
python -m pip install pylint
python -m pylint nthcommunity test/test_nthcommunity.py
```

CHAPTER

FIVE

CONTRIBUTIONS

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

CHAPTER

SIX

VERSIONING

Beginning with version 0.1.0, the version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.

CHAPTER

SEVEN

PUBLISHING

This library can be published as a package on PyPI by a package maintainer. Install the wheel package, remove any old build/distribution files, and package the source into a distribution archive:

```
python -m pip install wheel
rm -rf dist *.egg-info
python setup.py sdist bdist_wheel
```

Next, install the twine package and upload the package distribution archive to PyPI:

```
python -m pip install twine
python -m twine upload dist/*
```

PYTHON MODULE INDEX

n

nthcommunity.nthcommunity, 9

INDEX

C

collaboration (*class in nthcommunity.nthcommunity*), 9
contributor (*class in nthcommunity.nthcommunity*), 11
count (*class in nthcommunity.nthcommunity*), 12

E

encrypt() (*nthcommunity.nthcommunity.contributor method*), 11
evaluate() (*nthcommunity.nthcommunity.recipient method*), 10

F

from_json() (*nthcommunity.nthcommunity.collaboration static method*), 9
from_json() (*nthcommunity.nthcommunity.contributor static method*), 11

G

generate() (*nthcommunity.nthcommunity.recipient method*), 10

I

identifier() (*nthcommunity.nthcommunity.contributor method*), 11
integer (*class in nthcommunity.nthcommunity*), 13
intersection (*class in nthcommunity.nthcommunity*), 12

M

module
    nthcommunity.nthcommunity, 9

N

nthcommunity.nthcommunity
    module, 9

R

recipient (*class in nthcommunity.nthcommunity*), 9

S

ServiceError, 13
summation (*class in nthcommunity.nthcommunity*), 12

T

table (*class in nthcommunity.nthcommunity*), 13
to_json() (*nthcommunity.nthcommunity.collaboration method*), 9
to_json() (*nthcommunity.nthcommunity.contributor method*), 11

V

validate() (*nthcommunity.nthcommunity.contributor method*), 11