datastore_client Documentation

Release 0.0.1

Sam Mulube

Apr 04, 2019

Contents:

CHAPTER 1

Introduction

This library contains a minimal client library developed as part of the DECODE project in order to provide a Python client for the encrypted datastore component.

The purpose of this library is to enable a Python based client application to easily read from and write to the encrypted datastore.

The datastore API is implemented using a simple RPC framework called Twirp. This framework uses protocol buffers as a transport layer, and provides a suite of tools for generating server stubs as well as client bindings in a number of programming languages and can be thought of as being akin to a simpler version of GRPC but with fewer features and without the hard dependency on HTTP/2.

Because the client code is automatically generated from a set of protocol buffer definitions, some aspects of the API might not be perfectly idiomatic Python, but in general the affordances offered are fairly pythonic so should be relatively straightforward to use.

1.1 License

The library has currently been released under the Apache 2.0 License. This license is a permissive open source license that allows this code to be used freely in either open source or proprietary, closed source software.

1.2 Install

To install the library please run the following command:

```
$ pip install decode-datastore-client
```

CHAPTER 2

Usage

The client may be used to either read data from the encrypted datastore, or to write data via the two API methods the datastore exposes.

2.1 Initializing Client

A client instance can be initialized as follows:

```
from datetime import datetime, timedelta

from datastore_client.datastore_pb2_twirp import DatastoreClient

# create our API client
datastore_address = 'http://datastore:8080'

# initialize a client instance for the given datastore address
client = DatastoreClient(datastore_address)
```

Important: Currently the client does not require any authentication information to be be added when making requests, but this may change if the DECODE system requires this.

2.2 Reading Data

Here's a code snippet showing how the library may be used in order to read data from the encrypted datastore:

```
from datetime import datetime, timedelta

from datastore_client.datastore_pb2 import ReadRequest
```

(continues on next page)

(continued from previous page)

```
# we create a start_time of 1 hour ago
start_time = datetime.utcnow() - timedelta(hours=1)

# obtain the public key we are requesting data for
public_key = 'BGBgTKU7ZJHRBl...'

# construct our read request
read_request = ReadRequest()

# set the public key we are requesting data for
read_request.public_key = public_key

# set the start time from our datetime object
read_request.start_time.FromDatetime(start_time)

# set the page size we want to work with (max 1000)
read_request.page_size = 100

# now make the first request
response = client.read_data(read_request)

while True:
    for event in response.events:
        print(event.data) # encoded data requiring decryption
        print(event.event_time.ToJsonString())

    if response.next_page_cursor == "":
        break

    read_request.page_cursor = response.next_page_cursor
    response = client.read_data(read_request)
```

Note in the above code example we continue consuming data from the server until it returns an empty value for `next_page_cursor`. In a real application we would need to decode the retrieved data for each `event`.

2.3 Writing Data

Here's a code example showing how the client library may be used to write data:

```
from datetime import datetime, timedelta

from datastore_client.datastore_pb2 import WriteRequest

# obtain the public key we are writing data for
public_key = 'BGBgTKU7ZJHRBl...'

write_request = WriteRequest()
write_request.public_key = public_key
write_request.data = b'encrypted bytes'
write_request.device_token = 'abc123'

client.write_data(write_request)
```