DIMY COVID Contact Tracing System

Overview

DIMY COVID Contact Tracing Application application is a system implemented using DIMY protocol. [link here]. The whole system contains two parts, the front-end which runs on the client side, and the back-end which runs on an server side. The protocol implementation is based on a several full-strength general purpose cryptographic algorithms, such as ECDH, Shamir secret sharing, murmurhash3 etc.

The current implementation only contains the fron-tend part developed using Python3.7, and it depends on mmh3 and openssl libraries. The implementation has been tested on Linux environment and Python version 3.7 or above.

Notice

Terms: please refer to the spec!

Usage

First of all, please make sure you already have the required python libraries installed, the file requiresments.txt in the root source tree contains all the libraries installed on my system during developing. Not all of them are required, the main dependencies are base58(1.0.3), sslcrypto(5.3), pyaes(1.6.1) and mmh3(3.0.0). But if there is some dependencies issues, this is good place to have a look :D.

Tips for setting up the environment (Linux / MacOS) A good way to test the python program without messing up with the host environment is to create a virtual environment. To do this you can install venv.

1. create a virtual environment, python3 -m venv test-env
2. clone the source code to the virtual environment
    1. cd test-env
    2. git clone https://github.com/realprocrastinator/DIMY-grp.git
3. activate the virtual environment, source test-env/bin/activate
4. know install the dependencies, pip3 install -r ./DIMY-grp/src/requirements.txt
5. now you should be able to launch the program!

Once the required libraries has been installed, the main program can be started by run python3 main.py. Once the program starts, it will generate a template of default configuration file named default_config.json. By modifying this file, you can configure the program according to you taste. And to restart the program with the customised configuration, you need to use the -c flag. (e.g. python3 ./main.py -c my_config.json. The other flag named -n is used for debugging locally, if you only want to simulate the how does the whole

front-end behave running just one program instance. If running two or more instances, this flag can be omitted.

In the main loop, the user can feed in the command. Currently only two commands are supported, first is `s` to terminate the program. The second one is `c` to generate the `CBF` and upload to the specific back-end server and wait for reply.

Features

configuration

The default configuration file is auto generated into the location where the `main.py` is also this must be the root source file tree.

The default configuration file is a `json` file contains several entries as follow:

- `UDP_RCV_IP`: The IP address for the front-end app to listen to. By default it is set to `""`
- `UDP_RCV_PORT`: The IP port for the front-end app to listen to. By default it is set to `8080`
- `UDP_SND_IP`: The IP address for the front-end app to broadcast message. By default it is set to `255.255.255.255`
- `UDP_SND_PORT`: The IP port for the front-end app to broadcast message. By default it is set to `8080`
- `BF_BITS`: The default size of bits of each `BloomFilter` array. By default this is `800000` bits
- `BF_NHASHFUNS`: The default hash functions that will be used to insert into the `BloomFilter` by default this is `3`
- `BF_ARRSZ_BITS`: The default size of bits of each array inside of the `BloomFilter` array by default this is `8`
- `BFMGR_INIT_NDBFS`: The default number of `DBFs` will be initialized when the program starts
- `BFMGR_MAX_DBF_POOLSZ`: The maximum number of `DBFs` will be stored on each device during the program running
- `BFMGR_LOGLEVEL`: The default logging level of the `bfmanager's` logger. By default this is equal to `logging.DEBUG`
- `BGWRK_LOGLEVEL`: The default logging level of the `bgworker's` logger. By default this is equal to `logging.DEBUG`
- `BG_GEN_EphID_SECS`: The period of generating the `EphID` in seconds, by default this is `60`

- `BG_SHARE_EphID_SECS`: The period of sharing a part of the `EphID's` secret in seconds, by default this is `10`
- `BG_RECV_CHECK_SECS`: The period of checking the receiving buffer to see if can reconstruct the `EphID` periodically in seconds, by default this is `30`
- `BG_DBF_MOVE_TO_NEXT`: The period of updating the current `DBF` to the next one periodically in seconds, by default this is `600`
- `BG_DBFPOOL_UPDATE_SECS`: The period of generating new `DBF` in the `DBF` pool as well as deleting the old one if there is any existed longer than 1 hour periodically in seconds, by default this is `600`
- `BG_QBF_GEN_SECS`: The period of generating new `QBF` from the `DBF` pool as well as querying the the server in seconds, by default this is `3600`
- `URL_TEMPLATE`: The root `url` of the default back-end server:
  - `http://ec2-3-26-37-172.ap-southeast-2.compute.amazonaws.com:9000/comp4337/`
- `URL_SUFFIX`: The suffix of the default back-end server `url`

```
{
"UPLOAD" : "cbf/upload",
"QUERY": "qbf/query"
}
```

- `STDOUTF_LOGLEVEL`: The default logging level of the console output. By default this is equal to `logging.INFO`
- `ALL_LOG_LEVEL`: The default logging level of the all logger which can configure the module wide the individual logger. By default this is equal to `logging.DEBUG`
- `DEBUG_MODE`: The flag to turn on the debug mode, which will then enable, currently this is not used, as the debug log can be set using the `*_LOG_LEVEL` entry
- `ALL_LOG_FILE`: The default log file path with name, by default is `log.txt` under the root source director
- `NUM_SECRET_PARTS`: The default number of secret parts that `Shamir` algorithm will generate, by the fault is `6`
- `NUM_THRESHOLD`: The default minimal number of secret parts that `Shamir` algorithm require to reconstruct the secret, by the fault is `3`

logging

The logging subsystem has two output streams. One is the `stdout` and the other one is the `logfile` configured by the `ALL_LOG_FILE` entry.

Details about each module

Modules (TODO add details)

The entire front-end system contains:

- `bfmng` A storage component
- `bgwork` A background task scheduler
- `common` A communication module
- `config` A configuration helper module
- `idmng` A client `EphID`, `EncntID` management component
- `sslcrypto_client` A module ported from `sslcrypto` which implements the `ECDH` algorithm
- `test` A module contains several unit tests for the various modules
- `utils` A module contains several helper functions
- `tasks.py` A file contains all the background tasks wrappers
- `main.py` The entry point of the whole front-end system