DIMY COVID Contact Tracing System Report

Group number: 03

Member name & Zid: Jiawei Gao (z5242283), Siying Chen(), Yuntao Zhou()

Overview

This the front-end implementation of the DIMY COVID contact tracing system. The back-end is provided and hosted on AWS, the URL is included in the `default_conf.json` after running the program. The implementation is purely written in Python 3 and has been tested in both Linux, macOS, and Windows environments with the python version >= 3.7. The start the program with default parameters and configuration please go into the root source tree directory which contains `Dimy.py` and run `[python|python3] ./Dimy. py`. To make modifications to the configurations please refer to `README.md`.

Implementation Features

Task Managers

- According to the protocol, we have several tasks that need to be scheduled periodically, such as the EphID generating and secret sharing task, DBF management task, etc. Those tasks are implemented as background threads which will be triggered when the timer fires. In general, we for each task have a specific manager to manage it.
-
    - ID manager is responsible for generating the EphID using ECC every one minute and then uses the Shamir algorithm to generate 6 parts for the secret and broadcast them every 10 seconds.
    - Bloom Filter Manager is responsible for managing the DBF pool which has one DBF initially and continuously add a new DBF into the pool until the pool contains 6 DBFs. Besides, it will also delete the DBF that has a lifetime longer than 60 minutes and add a new one into the pool. The Bloom Filter manager will also combine the DBF pool which has 6 DBFs in into a QBF every 60 minutes of a CBF which is instead controlled by the user.
    - Background Task Manager is responsible for scheduling the background tasks that we listed above. Basically, it manages all the daemon threads and register a specific timer for each of them and then launch each thread.

Communication methods: Broadcasting and Receiving

- The implementation of broadcasting and receiving uses POSIX sockets. For each client application, one socket is used for broadcasting and the other one is used for receiving the broadcasting messages from other clients.
- The receiving method is implemented as a blocking daemon thread which checks the receiving buffer periodically. And the broadcasting method runs on another daemon thread, which broadcasts secret shares every 10 seconds.

Message Format

- The broadcasting message has its specific format as follow:

Length	**3 Bytes**	**1 Byte**	**16 Bytes**	**4 Bytes**
Field	Hash Tag	Section ID	Secret Part Payload	Sequence Number

- The Hash Tag is the first 3 bytes of the hash of the EphID calculated via sha256.
- Section ID indicates the order of the receiving parts, but the Shamir Algorithm doesn't require the ordering of the secret parts, so this field is just for counting purposes.
- Secret Part Payload is the secret part of the EphID generated using Shamir Algorithm.
- The Sequence Number field is our solution to the issue of the overlapping timing window. For example, let's define the initial time point as t0, client 1 sends the message at t0 + 30 seconds, and then client 2 starts to send its shared secret at t0 + 30 seconds as well. Then at time point t0 + 60 seconds, Ideally that two clients should generate Encounter ID, but since the privacy which used to generate the EphID of the client1 has already been changed. We can't reconstruct the same Encounter ID. To fix this problem, we save the previous privacy as well as the current privacy. and use the sequence number to determine which privacy should use, the old one or the new one. To make it simple, we set the length of this filed as 4 bytes, which is 32 bits long. Should be sufficient to run the program for a year.

logging

The logging subsystem has two output streams. One is the `stdout` and the other one is the `log.txt` configured by the ALL_LOG_FILE entry. The `stdout` is used for demonstration and the log file is used for diagnosing or inspecting the procedure.

configuration

The default configuration file is auto-generated into the location where the main.py is at, also this must be the root source file tree. The user of this program can modify the default configuration file and feed it to the program to meet their requirements.

Modules

The entire front-end system contains:

- `bfmng` A storage component
  - `bfmgr.py` implements the management methods of the DBF pool. Includes updating the DBF pool by adding a new DBF to it or removing one from it but never beyond 6 DBFs in the pool. Besides, it also implements the combining of the DBFs into a QBF/CBF by using the union method provided by the BloomFilter class.
  - `bloomfilter.py` implements the basic data structure of the BloomFlter class. The underlying implementation is using an array of unsigned char types and some basic algorithms for manipulating the data structure such as union or insert. By default, it uses `murmurhash3` as the hash function.
- `bgwork` A background task scheduler
  - `bgmgr.py` implements the background task scheduling method by maintaining a job queue, each of its entries contains a daemon thread.
  - `exception.py` implements the custom run-time error.
  - `job.py` implements the generic background task class.

- `common` A communication module
    - `udpmgr.py` implements the client-to-client EphID broadcasting and receiving methods using two dedicated sockets for multiplexing the message. This is just a wrapper of the python socket module.
    - `tcpmgr.py` implements the helper function for uploading the CBF to the server and querying the server with QBF.
    - `msg.py` implements the broadcasting message format class.
- `config` A configuration helper module
    - `configure.py` implements the default configuration and configuration parser.
- `idmng` A client `EphID`, `EncntID` management component
    - `idmgr.py` implements the EphID generating methods, Encounter ID generation methods and secret sharing methods, etc. This module is implemented on top of the `sslcrypto` module. The EphID is generated using ECDH algorithm. The
    - `shamir.py` is the implementation borrowed from Shamir Secret Sharing with some small modifications.
- `sslcrypto_client` A module ported from `sslcrypto` which implements the `ECDH` algorithm
    - A module borrowed from online implements the `ECDH` algorithm.
- `test` A module contains several unit tests for the various modules
    - Several unit tests against several custom implementations introduced above.
- `utils` A module contains several helper functions
    - `helper.py` implements some helper functions for handling and converting data type.
- `tasks.py` A file contains all the background tasks wrappers
    - All the background/non-background task wrapper functions go here.
- `Dimy.py` The entrance of the whole front-end system
    - The main entry of the program implements several bootstrap functions and the user command parser.

Challenges and Known Issues

The main challenges are:

- Since we have several threads running in the background, preventing them from facing raise conditions is not quite easy. Also to achieve the synchronisation, we used the `lock` and `conditional variable` which can potentially lead to deadlock issues.
- The overlapping timing issue introduced before leads to the inconsistent Encounter ID. Instead of discarding the package, we used store the previous privacy for generating the EphID using ECDH algorithm together with the sequence number in the message to choose whether to use to store old privacy or the new one.
- The testing can be time-consuming as the timing-parameters are relatively large. To achieve easy an debugging purpose, we separate the output stream to a log file and the console. Besides, we can manually configure the program's parameter's to simulate the behaviour but with a much shorter time interval.

Known Issues:

- The first minute of the broadcasting message can be received properly on some platforms.

Dinary

Contribution

Individual Part

- Jiawei is in charge of the following things **(1/3):**
    - Implementing the communication library between clients and servers using TCP protocol and Implementing the message protocol.
    - Implement a BF management library that can be used for managing DBF, CBF, and QBF.
    - implement the background timer module for firing the alarm periodically.
- Siying is in charge of the following things **(1/3)**:
    - implement the privacy generating library which handles the ephemeral ID generation.
    - Implement the Diffie-Hellman library.
    - implement the argument parser.
- Yuntao is in charge of the following things **(1/3)**:
    - Implement the "k out of n" sharing protocol.
    - Implement the message passing library based on UDP protocol.
    - implement the configuration parser module of this project.

Common Part

- Each member will take turns updating the diary.
- Each member will cooperate on the Github and maintain the source code and documentation together.
- Create the midterm presentation slides, final report, Demo video.
- Writing the test cases to find the flaws in the code.

Progress

Week	Completion	Issue
5	Create github repo, decide the main developing language and assign the presentation workload.	Questions about the DIMY protocol details, the format of diary and presentation and the wheels we can use.
6	Prepare presentation slides, record the video clips and assign tasks of implementation.	Questions about Video editing tool and Buckets to store videos.
7	Implement Background Timer Module and rapid prototype the framework.	Null.
8	Implement the helper modules and refine the framework.	Search for the appropriate libraries.
9	Implement ID management + Bloomfilter management module	Multi-threading programming designs.
10	Merge and refine the code + report	Null.