## SDK definition

# Initialization

1. initialize BLE service

2. initialize notification service

3. initialize WiFi service

# API

- (void)initialize: (void)device\_id {

// call function: generateSK

}

- (void)runBroadcast {

//enable BLE broadcasting services.

}

- (void)pauseBroadcast {

}

- (void)runReceiver {

//enable BLE receiver services.

}

- (void)pauseReceiver {

}

- (void)runEnvironmentScanning {

// enable wifi scanning services

}

- (void)pauseEnvironmentScanning {

}

- (void)contactNotification: (string)ntfContent {

// send notification content

}

- (void)uploadProfile {

//call function: - (string)readSK: and get secret key

// upload secret key to server

}

- (void)downloadProfile {

//call function: contactDetection:

}

# Direct Contact

- (void)startBLEScanning {

}

- (void)enterBLEScanningBackground {

}

- (void)endBLEScanning {

}

- (void)startBLEBroadcasting {

}

- (void)enterBLEBroadcastingBackground {

}

- (void)endBLEBroadcasting {

}

- (void)saveContactHistory {

//save UUIDs received by BLE receiver

}

- (void)generateSK:(string)device\_id {

// save device\_id as secret key locally

}

- (void)contactDetection:(String)downloadedProfile {

// call function: detectContactFrom:

// if have non zero return: call function contactNotification:

}

# EphID Generator, Processor

class EphID:

Flag: //for identification

Timestamp: //timestamp

SecretKey

class ContactHistory:

start\_timestamp

duration

- (string)generateEphID:(String)sk, (unsigned long long)timeStamp {

return EphID

}

- (Array)detectContactFrom:(string)secretKey, EphIDs:(Array)ephIds {

return an arrat of ContactHistory

}

- (string)readSK {

//return locally saved secret key

}

- (void)refreshEphID {

// Timer: per 3 hours call function: generateEphID

// save the new EphID locally.

}

- (void)fetchSK {

// fetch secrect keys from confirmed cases

// if a new secret key is downloaded, call function: contactDetection

}

# Indirect Contact

struct LOCATION

{

int timestamp;

map<string, double> scaned\_wifi;

};

//Given a location, we would like to detect whether a confirmed case has ever been to that location.

Class detection{

* **int cal\_risk(LOCATION w, list <LOCATION> trajectory);**

// given a location of user and the trajectory of a confirmed case,

//return -1 if the confirmed case has not been to the lcoation; return how many days before if the case has been to the location;

};

class RSSI\_encryption{

* **double add\_noise(double value);** // given the strength value of WIFI, the function add a gaussian to the value and return the new value;
* **string id\_encryption(string wifi\_id);** // given the WIFI ID, the function will encrypt it and return the encrypted result.

};