Android Contact Tracing API

Preliminary - Subject to Modification and Extension

April 2020

Version 0.4

Android Contact Tracing API

```
/**
 * Starts BLE broadcasts and scanning based on the defined protocol.
 *
 * If not previously used, this shows a user dialog for consent to start contact
 * tracing and get permission.
 *
 * Calls back when data is to be pushed or pulled from the client, see
 * ContactTracingCallback.
 *
 * Callers need to re-invoke this after each device restart, providing a new
 * callback PendingIntent.
 */
Task<Status> startContactTracing(PendingIntent contactTracingCallback);

@IntDef({...})

@interface Status {
  int SUCCESS = 0;
  int FAILED_REJECTED_OPT_IN = 1;
  int FAILED_SERVICE_DISABLED = 2;
  int FAILED_BLUETOOTH_SCANNING_DISABLED = 3;
  int FAILED_TEMPORARILY_DISABLED = 4;
  int FAILED_INSUFFICENT_STORAGE = 5;
  int FAILED_INTERNAL = 6;
}

/**
 * Handles an intent which was invoked via the contactTracingCallback and
 * calls the corresponding ContactTracingCallback methods.
 */
void handleIntent(Intent intentCallback, ContactTracingCallback callback);

interface ContactTracingCallback {
  // Notifies the client that the user has been exposed and they should
  // be warned by the app of possible exposure.
  void onContact();

  // Requests client to upload the provided daily tracing keys to their server for
  // distribution after the other user's client receives the
  // requestProvideDiagnosisKeys callback. The keys provided here will be at
  // least 24 hours old.
  //
  // In order to be whitelisted to use this API, apps will be required to timestamp
  // and cryptographically sign the set of keys before delivery to the server
  // with the signature of an authorized medical authority.
  void requestUploadDailyTracingKeys(List<DailyTracingKey> keys);

  // Requests client to provide a list of all diagnosis keys from the server.
  // This should be done by invoking provideDiagnosisKeys().
  void requestProvideDiagnosisKeys();
}

class DailyTracingKey {
  byte[] key;
  Date date; // Day-level granularity.
}
```

```
/**
 * Disables advertising and scanning related to contact tracing. Contents of the
 * database and keys will remain.
 *
 * If the client app has been uninstalled by the user, this will be automatically
 * invoked and the database and keys will be wiped from the device.
 */
Task<Status> stopContactTracing();
```

```
/**
 * Indicates whether contact tracing is currently running for the
 * requesting app.
 */
Task<Status> isContactTracingEnabled();
```

```
/**
 * Flags daily tracing keys as to be stored on the server.
 *
 * This should only be done after proper verification is performed on the
 * client side that the user is diagnosed positive.
 *
 * Calling this will invoke the
 * ContactTracingCallback.requestUploadDailyTracingKeys callback
 * provided via startContactTracing at some point in the future. Provided keys
 * should be uploaded to the server and distributed to other users.
 *
 * This shows a user dialog for sharing and uploading data to the server.
 * The status will also flip back off again after 14 days; in other words,
 * the client will stop receiving requestUploadDailyTracingKeys
 * callbacks after that time.
 *
 * Only 14 days of history are available.
 */
Task<Status> startSharingDailyTracingKeys();
```

```
/**
 * Provides a list of diagnosis keys for contact checking. The keys are to be
 * provided by a centralized service (e.g. synced from the server).
 *
 * When invoked after the requestProvideDiagnosisKeys callback, this triggers a
 * recalculation of contact status which can be obtained via hasContact()
 * after the calculation has finished.
 *
 * Should be called with a maximum of N keys at a time.
 */
Task<Status> provideDiagnosisKeys(List<DailyTracingKey> keys);

/**
 * The maximum number of keys to pass into provideDiagnosisKeys at any given
 * time.
 */
int getMaxDiagnosisKeys();
```

```
/**
 * Check if this user has come into contact with a provided key. Contact
 * calculation happens daily.
 */
Task<Boolean> hasContact();

/**
 * Check if this user has come into contact with a provided key. Contact
 * calculation happens daily.
 */
Task<List<ContactInfo>> getContactInformation();

interface ContactInfo {
  /** Day-level resolution that the contact occurred. */
  Date contactDate();

  /** Length of contact in 5 minute increments. */
  int duration();
}
```